Project Number:	**IST-1999-10077**
Project Title:	**AQUILA** **Adaptive Resource Control for QoS Using an IP-based Layered Architecture**
Deliverable Type:	**PU - public**

Deliverable Number:	**IST-1999-10077-WP2.2-TUD-2203-PU-R/b0**
Contractual Date of Delivery to the CEC:	**December 31, 2001**
Actual Date of Delivery to the CEC:	**December 21, 2001**
Title of Deliverable:	**User Guide for End-user Application Toolkit**
Workpackage contributing to the Deliverable:	**WP 2.2**
Nature of the Deliverable:	**R – Report**
Editor:	**Falk Fünfstück (TUD)**
Author(s):	**Andreas König (BAG), Sotiris Maniatis, Haris Tsetsekas (NTU), John Karadimas (QSY), Falk Fünfstück, Anne Thomas (TUD)**

Abstract:	This deliverable D2203 reports on the application and user interfaces, which the EAT provides for the second AQUILA trial. These are: the QoS API, the QoS GUI/Portal, the Application Profiles, and the Application Level Gateways. The deliverable describes in detail their functionality and shows usage scenarios.
Keyword List:	AQUILA, IST, User Guide, Developer Guide, Application Interfaces, QoS API, User Interfaces, GUI, QoS Portal, Application Profiles, Legacy Applications, Basic Internet Applications, Application Level Gateways, Complex Internet Services

Executive Summary

This deliverable describes the interfaces of the End-user Application Toolkit (EAT) towards the applications and users of the second AQUILA trial.

"The modular End-user Application Toolkit will be made available in time to be used in the development of the applications for the second trial. As a working document for the development of the end user applications for the second trial, this deliverable will provide a detailed guide of how to apply the End-user Application Toolkit, including a full explanation of the software architecture used for application services. Target audience for this deliverable are the developers of end-user applications." [Technical Annex]

This deliverable therefore focuses on the detailed description of the functionalities offered by the EAT. This includes also some scenarios on how to use the functionalities for so-called Basic Internet Applications as well as for Complex Internet Services.

One of the outcomes of the first trial is that Basic Internet Applications (here also called legacy applications) can be supported by the AQUILA architecture by providing different mechanisms such as Graphical User Interfaces (GUIs), Application Profiles, and Protocol Gateways (her also called Application Level *Proxies*). However, this support needs the involvement of (human) customers (end-users), because legacy applications cannot be modified in order to be QoS-aware.

Consequently, this deliverable also includes the explanation of the *user* interfaces the EAT offers. Moreover, user interfaces are an essential part of modern (complex) Internet services that provide multimedia services towards Internet users.

In that way, the application and user interfaces, developed for the first trial, can – with some modifications – be reused for the applications of the second trial. More specifically, the application and user support will be based on the following main "pillars":

- An Application Programming Interface (API) for QoS-aware applications,
- Graphical User Interface for manual reservation in different modes,
- Application Profiles to allow QoS mapping from a technical to a user-friendly level, and
- Application Level Proxies to support some special application signalling protocols.

The one responsible middleware that offers all these interface is the EAT which acts as the only QoS portal to the AQUILA's RCL controlled network.

Table of Contents

1 INTRODUCTION 7

2 APPLICATION INTERFACES 8

2.1 APPROACH 8

2.2 EAT API 10

2.2.1 Description of Functionality 10

2.2.2 Usage Scenario 16

2.3 GRAPHICAL USER INTERFACES – THE AQUILA PORTAL 18

2.3.1 Login 18

2.3.2 Advanced and Regular Reservation Requests 19

2.3.2.1 Regular Reservation GUI 19

2.3.2.2 Advanced Reservation GUI 22

2.3.3 Active Reservations 24

2.3.4 Logout 26

2.3.5 Navigation/Appearance 27

2.4 APPLICATION PROFILES 28

2.4.1 Definition Application Profiles 28

2.4.2 Specification of Application Profiles at Application Level 30

2.4.3 Specification of Application Profiles at Network Level 31

2.4.4 Utilisation 33

2.5 PROXIES 34

2.5.1 Use of the Proxies 34

2.5.1.1 GUI-initiated Reservation 34

2.5.1.2 Proxy-initiated Reservation 35

2.5.2 Implementation of a new Proxy 35

2.6 EAT SCRIPT 38

2.6.1 Script Specification 39

2.6.1.1 Document Type Definition 39
2.6.1.2 XML Examples 40
2.6.2 Command Line Tool 40
3 APPLICATION SCENARIOS 42
3.1 BASIC INTERNET APPLICATION SUPPORT 42
3.1.1 User-initiated Approach 43
3.1.2 EAT-initiated Support 43
3.1.3 Reservation Units and Groups 44
3.2 COMPLEX INTERNET SERVICE SUPPORT 44
3.2.1 Recommendations for the Creation of QoS Level 45
3.2.2 Recommendations for the Creation of a Set-up Menu for the QoS Service 46
3.2.3 Recommendations for the Interaction of the CIS and the EAT 49
3.2.4 Recommendations for the Feedback on the Status of the Connection of the QoS Categories 52
4 CONFIGURATION GUIDE FOR EAT 54
4.1 EAT SETTINGS 54
4.2 PROXY SETTINGS 55
5 ABBREVIATIONS 56
6 REFERENCES 59
APPENDIX 60
THE APPLICATION PROGRAMMING INTERFACE 60
api.idl 60
SampleClient.java 70
APPLICATION PROFILE 74
ApplicationProfile.dtd 74
CodecSpecification.dtd (Codec Profile) 75
Example: Codec G_723_1.xml 78
EAT SCRIPT 80
EATScript.dtd 80

Example.xml 81
Example1.xml 84
EAT & PROXY CONFIGURATION 84
EATSettings.dtd 84
Example: eat_dresden.xml 84
ProxySettings.xml 84
Example: sip_dresden.xml 85

Table of Figures

FIGURE 2-1: THE DIFFERENT APPLICATION INTERFACES OF THE EAT 9
FIGURE 2-2: APPLICATION AND USER INTERFACES OF THE EAT 10
FIGURE 2-3: THE EAT API (1) 11
FIGURE 2-4: THE EAT API (2) 15
FIGURE 2-5 THE LOGIN GUI 19
FIGURE 2-6: THE MENU OF THE AVAILABLE APPLICATIONS 20
FIGURE 2-7: THE ADVANCED RESERVATION GUI 22
FIGURE 2-8: THE REGULAR RESERVATION GUI 22
FIGURE 2-9: THE ACTIVE RESERVATIONS GUI 26
FIGURE 2-10 :THE AQUILA PORTAL – THE MAIN MENU AND THE REGULAR RESERVATION GUI 28
FIGURE 2-11 : EXAMPLE OF THE ARCHITECTURE IN THE COMPLEX INTERNET SERVICE SCENARIO SHOWING THE COMBINATION OF NON-QOS AWARE BASIC INTERNET APPLICATIONS AND APPLICATION PROFILES 29
FIGURE 2-12: APPLICATION ANALYSIS DIAGRAM AND THE ACCOUNTABILITY KNOWLEDGE LEVEL PATTERN 31
FIGURE 2-13: ANALYSIS UML CLASS DIAGRAM OF THE APPLICATION AT APPLICATION AND NETWORK LEVEL 32
FIGURE 3-1: SETTINGS: USER SELECTION 46
FIGURE 3-2: SETTINGS: CONNECTION EXPERT 47
FIGURE 3-3: SETTINGS: CONNECTION NOVICE 47
FIGURE 3-4: SETTINGS: QOS CATEGORY EXPERT 48
FIGURE 3-5: SETTINGS: QOS CATEGORY NOVICE 48
FIGURE 3-6: QOS PROFILE MEDIAZINE 49
FIGURE 3-7: SETTINGS: ALL QOS CAT. EXP. USER 50
FIGURE 3-8: APPLICATION PROFILE REALPLAYER 51
FIGURE 3-9: CORRELATION BETWEEN QOS PROFILE AND APPLICATION PROFILE 52

Table of Tables

TABLE 2-1: TYPICAL SCENARIO FOR THE USAGE OF THE EAT API 18
TABLE 2-2: TRAFFIC DESCRIPTOR AND POSSIBLE VALUES 32
TABLE 3-1: MEDIAZINE CONNECTION SETTINGS 47
TABLE 3-2: MEDIAZINE QOS CATEGORY SETTINGS 48
TABLE 3-3: RECOMMENDATIONS FOR MEDIAZINE USERS 53

1 Introduction

This deliverable gives an overview of the application and end-user interfaces the EAT offers in the second AQUILA trial.

The purpose of these interfaces is:

- To support the developers of new QoS-aware applications and services by providing an Application Programming Interface (API) on the top of the EAT which is a QoS middleware between applications and the (AQUILA) network.
- To support the users of existing, QoS-unaware legacy applications by providing Graphical User Interfaces (GUIs) for manual QoS requests in different usage modes. Since the most users might be not familiar with AQUILA, the flexible approach of predefined Application Profiles is used to allow a mapping between user's QoS understanding and the technical details of AQUILA QoS reservation requests.
- To support some special applications based on dynamic signalling protocols by providing transparent Proxies (Gateways) which collect important information directly from application's traffic in order to use them for reservation requests..
- To support also automatic tests and measurements by providing a script interface for the batch processing of QoS requests.

By offering all these different interfaces towards various, legacy and new applications, services as well as real end-users, the EAT aims to be a powerful and flexible QoS portal towards the Resource Control Layer. For legacy applications, this has already been proved in the first trial. The second trial, moreover, will show how the EAT can be applied by QoS offering Complex Internet Services, for example.

The deliverable is structured as follows:

After the description of the overall approach useful for getting familiar with the EAT, the following chapter 2 describes the several, above mentioned application interfaces in more detail. The reader may select only one chapter he/she is interested in. Note that the chapters focus mainly on the functionality the interfaces offer.

More impressive scenarios on how to use them are given in chapter 3 which is split into two parts: one for the usage of Basic Internet Applications, and the other showing how Complex Internet Services can benefit from QoS.

After that, a short installation and configuration guide of the EAT is given in chapter 4.

The document ends with an Appendix containing detailed interface specifications, examples, etc.

2 Application Interfaces

2.1 Approach

The End-user Application Toolkit for the second trial supports two major kinds of (Internet) applications:

- Legacy Applications [D1201] that are in fact QoS-unaware and that cannot be modified in order to directly access the EAT or any other QoS infrastructure. The most of existing Internet applications are legacy ones. The EAT prototype for the first trial already supported a set of them.

- QoS-aware Applications [D1201] that can themselves request for QoS, by using an API, for example (EAT-based Applications use the EAT API), or by using signalling protocols such as RSVP and SIP.

Internet applications, however, have also to be distinguished with regard to their *complexity*. In AQUILA, we make a distinction between Basic Internet Applications and Complex Internet Services [D1202]. They have to be supported in different ways: Whereas Basic Internet Applications are often legacy ones which cannot directly use the EAT, Complex Internet Services can be QoS-aware or even EAT-based although they consist of basic applications. It is one of the aims of the second trial, to demonstrate this feature as it is described in this document.

Generally, the EAT provides – at the control plane – a set of application interfaces in order to support the wide range of different applications (Figure 2-1):

- Legacy applications do not interact with the EAT. QoS reservations must therefore be made manually (see below).

- For some specific legacy applications that dynamically negotiate data port numbers, a special Proxy (Protocol Gateway) (e.g. for H.323) detects the for an AQUILA reservation request necessary information from the application's data flow [D2201].

- For applications that rely on signalling protocols, Protocol Gateways (Proxies) are provided that interpret protocol messages and map them to AQUILA reservation requests. For the second trial, a SIP Proxy will be realised. However, the Proxy Framework is flexible and extensible in order to include additional Proxies (e.g. for RSVP) later on. (More details can be found in chapter 2.5.)

- For EAT-based applications, a special Application Programming Interface (API) provides interfaces and methods for login, reservation requests and releases, etc. This proprietary API is accessible via CORBA and provides the full AQUILA functionality. (More details can be found in chapter 2.2.)

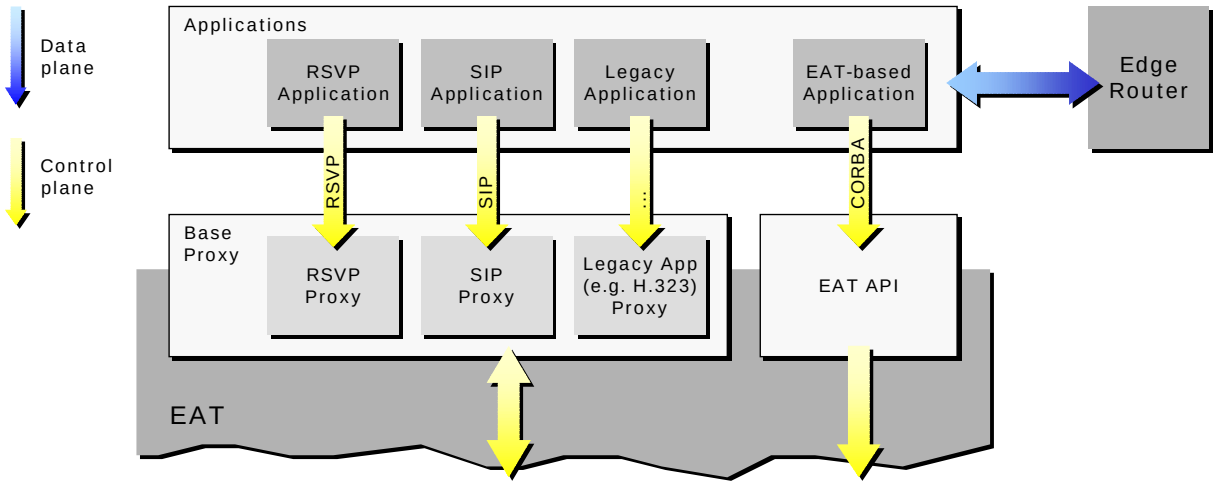

Figure 2-1: The different application interfaces of the EAT

Due to the fact that the EAT is fully transparent for legacy applications – even if they are supported by a Proxy – QoS reservations must be performed in a different way. For that reason, the EAT provides a set of graphical user interfaces in form of Web pages, in which an end-user can *manually* request for QoS reservations (Figure 2-2). Moreover, the so-called AQUILA Portal offers among other things two different reservation modes: an advanced one for end-users that have knowledge about the technical details of an AQUILA request, and a regular one for end-users that are not familiar with AQUILA. (More details can be found in the chapter 2.3.)

In order to support the regular reservation mode, an additional application "interface" is provided, the so-called Application Profile methodology [D2202]. Application Profiles contain reservation "schemes" with technical parameters mapped to well understandable QoS metaphors. (More details can be found in the chapter 2.4)

Note that the regular reservation mode is not necessarily part of the AQUILA Portal. In fact, Application Profiles are usable via the EAT API and can therefore be called by every Complex Internet Service that wants to make use of the AQUILA's QoS capabilities. In that way, such an Internet service may offer its own regular reservation mode, by showing the QoS metaphors from the proper Application Profiles of its basic applications/plug-ins.

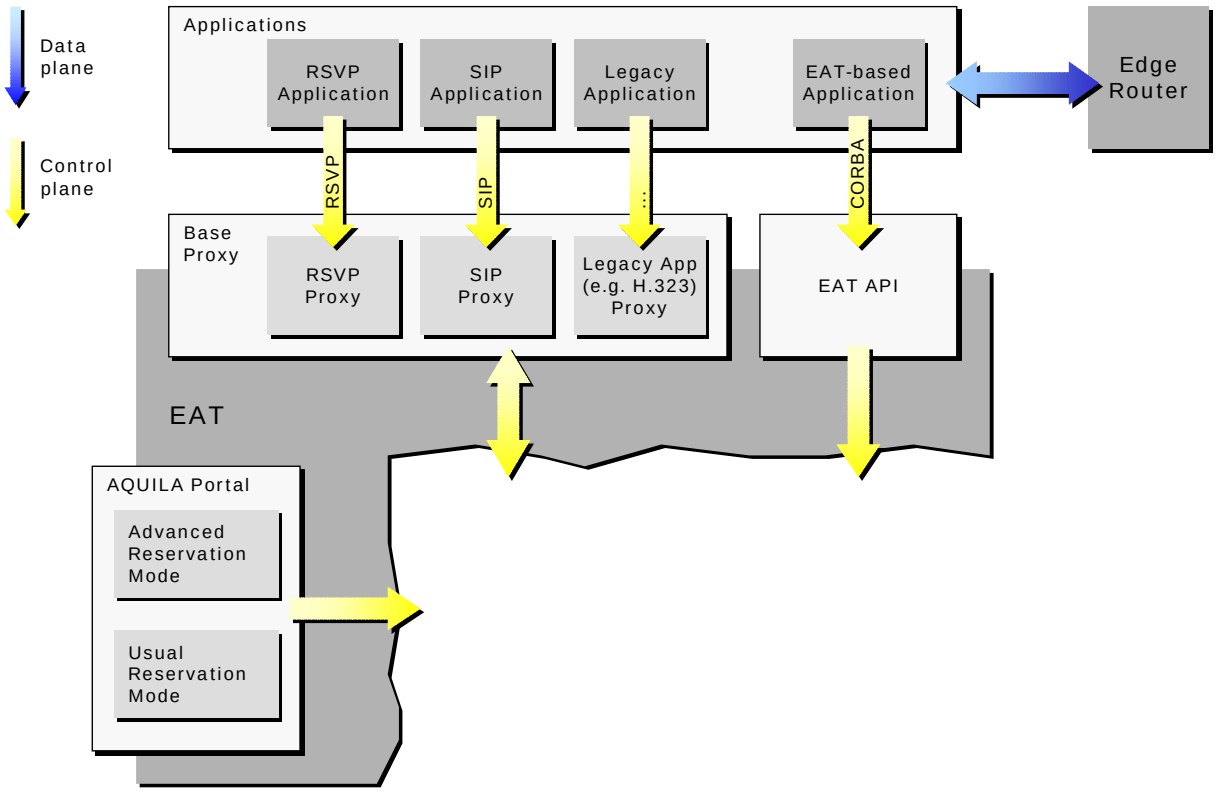

Figure 2-2: Application and user interfaces of the EAT

2.2 EAT API

2.2.1 Description of Functionality

The EAT API, in general, provides as set of CORBA interfaces and supporting data types towards applications and services in order to allow them the access to the AQUILA's network services and reservations.

This chapter gives an overview on the available interfaces, operations, structure types, and attributes. Moreover, instructions are given on how to use them. (The full interface specification is given in IDL in the Appendix.)

The following first UML class diagram depicts the API model with the most important interfaces and classes that belongs to users and reservations[1]. A detailed description follows.

[1] For all types and interfaces, please, see the api.idl in the Appendix.

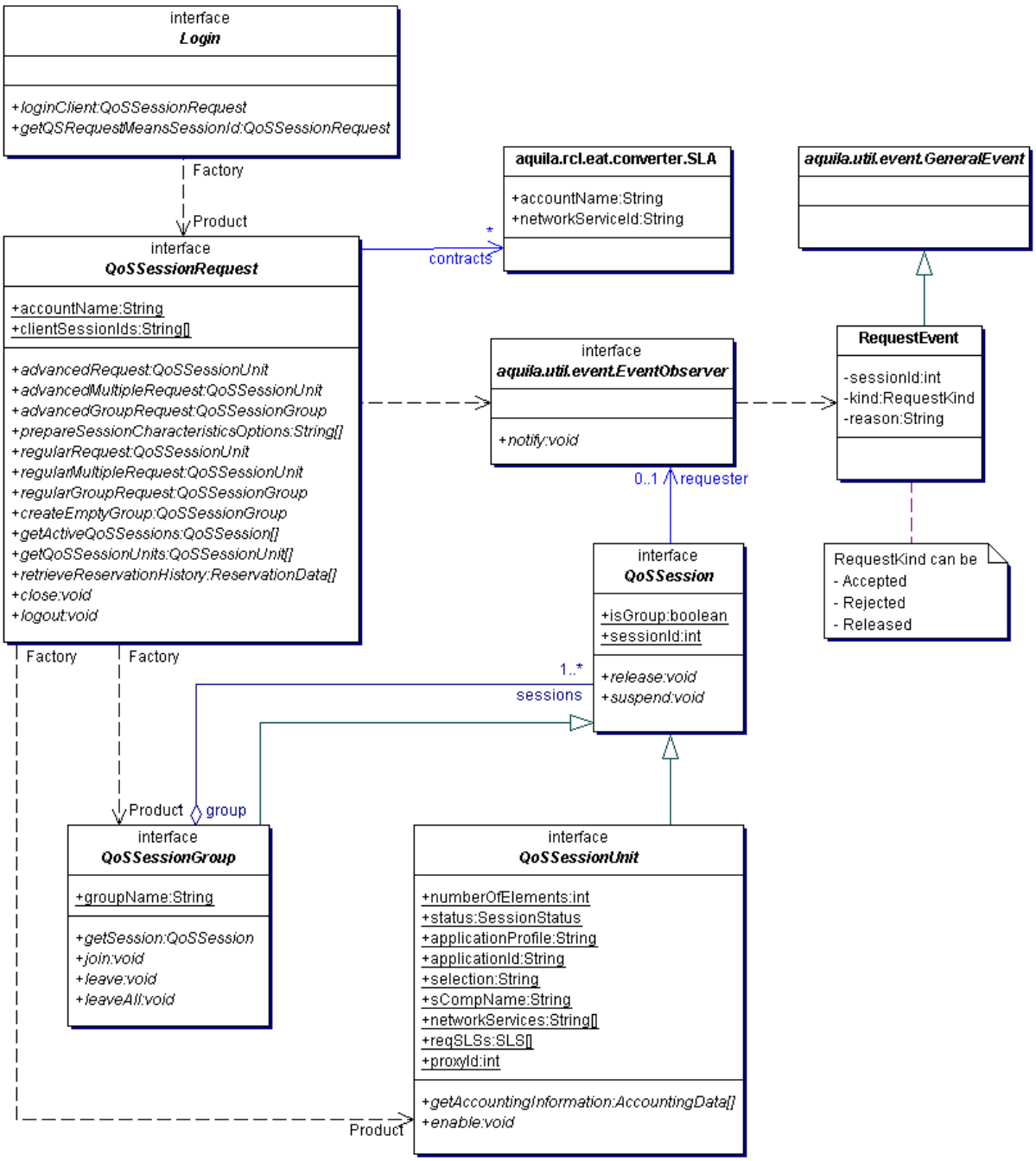

Figure 2-3: The EAT API (1)

Description:

The interface **`Login`** is implemented by a singleton object, to be used by all clients of this EAT (see paragraph 2.2.2). It provides:

- The `loginClient()` operation with two parameters: the login information (including user account and password) and an *optional* session id of the client (usually a web session id). `loginClient()` returns – if successful – the reference to an object of the interface type `QoSSessionRequest` which belongs to the now logged end-user. If an end-user severally logs in at the same time, he/she always gets the same reference. If something goes wrong with the login, an `APIException` is raised including a string with the reason.

- The `getQSRequestMeansSessionId()` operation with one parameter: the client session id of a previous call of the `loginClient()` operation. By using this id, it is possible to associate and return the reference to the belonging `QoSSessionRequest` object without to login once again. That is useful if a (web) client is not able to store the reference it gets from the previous call.

The interface **`QoSSessionRequest`** is the main user agent for a QoS requesting client. For each logged end-user exactly one object exists that implements this interface. It offers the following:

- Three readable attributes: The account name of the logged user, its contracts (namely its **`SLA`**s), and the associated client session ids (at most five).

- The `advancedRequest()` operation with two parameters: the request spec of the type `AdvancedSpec` including for example network service id and SLS, and the *optional* requester (see interface `EventObserver`), usually the requesting client. If a reservation can be established, the operation returns the reference to a new object of the interface type `QoSSessionUnit` containing one single reservation/session element (e.g. for a unidirectional reservation). Otherwise, an `APIException` is raised.

- The `advancedMultipleRequest()` operation. Instead of requesting a single, unidirectional reservation, a multiple (e.g. a bi-directional) reservation is requested. The request parameters can only differ from the requested services and the service level specifications. An inseparable `QoSSessionUnit` is created and returned, containing as many single elements as requested (e.g. two for a bi-directional reservation).

- The `advancedGroupRequest()` operation for different reservation requests within a group, for example for several service components per application, or for several applications per Complex Internet Service. A `QoSSessionGroup` is created and returned. (The group may get its own group name.)

- The `prepareSessionCharacteristicsOptions()` requests for the available options for a reservation in the regular mode in order to present them to the end-user. The only parameter indicates which Application Profile is to be used for that (see interface `ApplicationManager`).

- The `regularRequest()` operation with again two parameters: one of the type `RegularSpec` including for example the Application Profile, one service component and the chosen QoS option, and one for the requester. A `QoSSessionUnit` is created and returned.

- The `regularMultipleRequest()` operation like above but for multiple reservations (see `advancedMultipleRequest()`). The single reservation elements can only differ from the flow specs and scopes, for example regarding the source and destination addresses.

- The `regularGroupRequest()` operation for multiple groups, for example for the different service components of one application.

- The `createEmptyGroup()` operation to establish an empty `QoSSessionGroup`, optionally with a meaningful name.

- The `getActiveQoSSessions()` operation to return all active sessions (groups and units) of this client/end-user.

- The `getQoSSessionUnits()` operation to get all active session *units* of this client/user in a uni-dimensional array.

- The `retrieveReservationHistory()` operation to have a full list of all accepted former and actual reservations including their parameters and accounting information.

- The `close()` operation with the client session id as the only parameter, in order to de-associate a client (web) session from this object.

- The `logout()` operation to log off the end-user and to *release* all of his/her reservations (in contrast to `Login.close()`).

The interface **`QoSSession`** will not directly be implemented but is to abstract from its children `QoSSessionGroup` and `QoSSessionUnit`, and to have a common return type. Nevertheless, it provides the following attributes and abstract operations:

- Four attributes: `sessionId` as the *unique* session identifier, `isGroup` to indicate whether the session is a group or a unit, `group` to refer to the aggregating group if one exists, and `requester` to refer to the optional requesting client.

- The `release()` operation to release this session (unit or group) including all of its elements (groups, units, and/or single elements). Accounting data are automatically requested and stored within the EAT.

- The `suspend()` operation to cancel either this session (unit or group) without storing any accounting data.

The interface **`QoSSessionGroup`** as specialisation of `QoSSession`. It is for reservation groups which have the purpose to be better "manageable" than many separate reservations (e.g. to suspend the whole group). The interface provides the following features:

- The attribute `groupName` to show the purpose of the group.
- The attribute `sessions` to refer to the member `QoSSessions` of this group. Thus, these can either be `QoSSessionUnits` or again `QoSSessionsGroups`.
- The `getSession()` operation with one parameter: the id of a member session. It returns the reference to this object.
- The `join()` operation with one parameter: the reference to an already established session unit or group in order to include it into this group. By using this, multidimensional groups can be built.
- The `leave()` operation with the same parameter in order to quit the membership of the belonging session unit or group in this higher-level group.
- The `leaveAll()` operation to quit the membership of *all* session units or groups of this higher-level group. After that, this group can be released without releasing its members.

The interface **`QoSSessionUnit`** provides access to an inseparable reservation unit consisting of one ore more reservation elements. Therefore, the features are:

- The attributes `isMultiple` and `numberOfElements`. The first flag indicates whether the session unit is a multiple one, the second attribute gives the number of the reservation elements.
- The attribute `status` containing the session status. It can be `Provisional` (waiting for the Proxy information), `Active` (established in the ACA but not in the edge router), or `Enabled` (established also in the edge router).
- A list of attributes that belong to this session unit, e.g. the requested services and SLS.
- The `getAccountingInformation()` operation to explicitly ask for the last accounting status. Note, that the accounting data are automatically requested and stored when the reservation is released.
- The `enable()` operation to establish an already activated session also in the edge router reservation.

The interface **`EventObserver`** should be implemented by a client that wants to be informed when something happens with the reservation. For example, a reservation request that relies

on the support of a Proxy results in a session which is still *provisional* until the Proxy notifies the EAT that the reservation can really be activated:

- The `notity()` operation in this case informs the requesting client about this event, which is of the type **`RequestEvent`**. And due to the fact that the reservation request towards the *network* can fail even if it is provisionally established within the EAT, the requester is informed about the success of the request (`Accepted` or `Rejected`). Another important event occurs, when an already established reservation is automatically `Released` if the connection to the network fails. For that reason, `RequestEvent` contains one of these three kinds of events as well as the id of the belonging `QoSSession` and an optional string for more details on the reason.

The following second part of the API model depicts other supporting classes and interfaces:

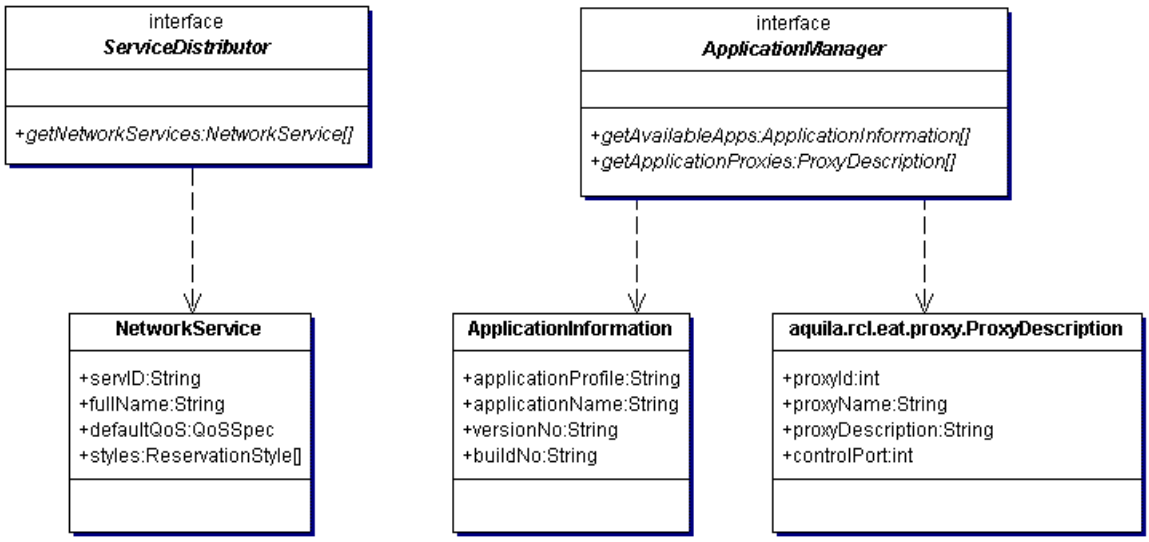

Figure 2-4: The EAT API (2)

Description:

The interface **`ServiceDistributor`** implemented by a singleton object provides access to the existing network services:

- The `getNetworkServices()` operation returns an array of the available network services including their specific values for QoS, reservation styles, etc.

The interface **`ApplicationManager`** also implemented by a singleton object is for the retrieval of the existing Application Profiles and the installed Proxies:

- The `getAvailableApps()` operation returns an array of **`ApplicationInformation`**, namely the identifiers of the profiles as well as the names, the versions, and optionally the

build numbers of the applications for which profiles exist. It is for the regular reservation mode.

- The `getApplicationProxies()` operation returns an array of the installed application Proxies containing in **`ProxyDescription`** their ids, names, etc. It is mainly for the advanced reservation mode (in order to select a Proxy) but also for the regular mode (in order to check the availability of the Proxy specified in the Application Profile).

2.2.2 Usage Scenario

A. How to access the CORBA API

The EAT API is a CORBA one specified in IDL and using the JDK 1.3 ORB. In order to get access to it, you have to do the following first:

1. If available, read the property file (e.g. aquila.rc) including the basic ORB parameters: "ORBInitialHost" and "ORBInitialPort". Otherwise these parameters are required at the command prompt when your start your client.
2. Initialise the ORB with these parameters.
3. Narrow the naming service called "NameService" and get the reference to it.
4. Create the initial name context "aquila.rcl" and retrieve the reference to it.
5. Get the name of the EAT from command line.
6. Narrow the following singleton objects and get references to them:
   - `Login`
   - `ServiceDistributor`
   - `ApplicationManager`

B. How to use the API

After getting the reference to the Login object, a typical scenario is the following:

Step		Operation
1	Login.	`Login.loginClient()`
1a	Get the reference to your QoSSession-Request agent either directly or via	`Login.getQSRequestMeansSessionId()`
2	Retrieve information on the existing network services, …	`ServiceDistributor.getNetworkServices()`

	… on the installed Proxies, …	`ApplicationManager.getApplicationProxies()`
	… and on the available apps.	`ApplicationManager.getAvailableApps()`
3	Get the SLAs	`QoSSessionRequest.contracts()`
4a	Request in the advanced reservation mode for a single reservation (unit), …	`QoSSessionRequest.advancedRequest()`
	… for a multiple reservation unit, …	`QoSSessionRequest.advancedMultipleRequest()`
	… or for a reservation group.	`QoSSessionRequest.advancedGroupRequest()`
4b	Prepare the options for the regular reservation mode.	`QoSSessionRequest.prepareSessionCharacteristicsOptions()`
	Request for a single reservation for *one* service component of the app, …	`QoSSessionRequest.regularRequest()`
	… for a multiple reservation unit, …	`QoSSessionRequest.regularMultipleRequest()`
	… or for a reservation group.	`QoSSessionRequest.regularGroupRequest()`
4c	Get the reference to the established reservation session(s) either directly or via	`QoSSessionRequest.getActiveQoSSessions()`
5	If a Proxy is required for the above reservation request, be prepared to *receive* an “accepted” or “rejected” event.	← `EventObserver.notify()`
6	Create an empty reservation group.	`QoSSessionGroup.creatyEmptyGroup()`
7	Perhaps join an already established session to a group, …	`QoSSessionGroup.join()`
	… or leave from it.	`QoSSessionGroup.leave()`
8	Perhaps retrieve the accounting data of a single reservation (unit).	`QoSSessionUnit.getAccountingInformation()`
9	Release a reservation group *without* releasing its members.	`QoSSessionGroup.leaveAll();` `QoSSessionGroup.release()`
10a	Release a reservation unit or group.	`QoSSession.release()`
10b	Or just suspend (cancel) it.	`QoSSession.suspend()`
10a	Close the client’s session (without releasing any reservations).	`QoSSessionRequest.close()`

10b	Or hardly logout (*with* releasing all your reservations).	`QoSSessionRequest.logout()`

Table 2-1: Typical scenario for the usage of the EAT API

Note that many of the operations above can raise exceptions of the type `APIException`. You have to catch them. For more details, please, see the example in the Appendix of this document.

2.3 Graphical User Interfaces – The AQUILA Portal

Taking into account the classification of the applications supported by the EAT (see chapter 2.1), the AQUILA Portal (portal) provides a set of graphical user interfaces (GUI) to allow the end-users to interact with the EAT, in order to ask for QoS reservations on behalf of legacy applications. In other words, the end-user is responsible to manually set-up and release reservations for such an application, asynchronously to the application.

The GUI has the form of Web pages, so the only software program the user has to install is one of the standard Web browsers. The user has also to be provided with the URL of the portal's first page, which will be something like: `http://<portal-host>/AQUILA.jsp`. Navigation among the portal's GUI pages is performed automatically, so the user does not have to know any other URLs.

In a few words, the following scenario is foreseen: the user first performs a login operation, so that the EAT can authenticate the user. If the login operation is successful, the user can interact with the portal to perform reservations and releases of established reservations, or to manipulate the active reservations into groups. At the end, the user can perform a logout operation. The actual intended application could be launched just after (or before) the user has placed a reservation.

The portal is intended for people that are "regular" users (AQUILA inexperienced users) without knowledge of technical QoS issues, as well as for computer or network professionals (AQUILA experienced users) that may have deep knowledge of the QoS requirements of the application they use, develop and test, as well as of the corresponding network mechanisms. So, the portal offers two basic reservation modes: a Regular one for the former kind of users, and an Advanced one for the latter.

2.3.1 Login

The first operation to be performed before a user can use the portal is the "login" operation. This operation performs user authentication with parameters the user's login-name and the password. So the portal's first page offers two fields that the user has to fill in: the 'Login' field and the 'Password' field.

After successful authentication, the user can perform all the other operations offered by the EAT. Moreover, in order to forbid the direct loading of a GUI page without the user to be authenticated, the EAT always checks whether the user is logged in or not. In case the user has not logged in, the portal notifies the user and proposes to load the login page.

If the authentication fails, the portal loads a similar page with the Login and Password fields, notifying the user that the authentication has failed.

The login page is the first page to be displayed when a user connects to the AQUILA Portal and is shown in Figure 2-5.

Figure 2-5 The Login GUI

2.3.2 Advanced and Regular Reservation Requests

It was mentioned before that the portal supports both regular users as well as advanced ones, by providing the reservation GUI in two modes: a regular one and an advanced one. These modes are discussed below.

2.3.2.1 Regular Reservation GUI

The Regular Reservation mode facilitates the placement of a reservation request, by exploiting the existence of some technical developments within the EAT, that are the Application Pro-

files and the Converter[2]. The fundamental point is that the Regular Reservation has the *knowledge* about specific session characteristics that are comprehensible by a regular user.

The EAT has defined these technical mechanisms in such a way that an Application Profile can be constructed and stored in the persistence layer of the EAT. The stored Application Profiles give a clue to the EAT about which applications are currently *available* in this mode. So, the first operation to be performed when the user selects the Regular Reservation GUI is to display the menu of the available applications (see Figure 2-6).

Figure 2-6: The menu of the available applications

After the user has selected the desired application, the specific GUI is constructed and displayed, presenting to the user the available *Options* for the selected application (see Figure 2-8).

The application may consist of more than one *component*, e.g. a multimedia application usually has an audio session, a video session, and (more than one) data sessions. The GUI page will contain all the available options for each component separately. This enables the users to perform reservations only for one, some or all the components, according to their requirements. Figure 2-8 presents an example page for the NetMeeting application, which gives options for two components: Audio and Video. There are actually three options for the Audio components, and four options for Video.

[2] The Converter is a logical component of the EAT that calculates QoS reservation parameters by using the Application Profiles.

All options are constructed of *session characteristics* that are comprehensible to the "regular" user. As an example, the session characteristics for Video are 'Video Quality', 'Window Size', and 'Network Speed'. Each session characteristic is described by a *qualifier* that identifies the actual offering of this session characteristic for each option. The qualifiers for the 'Video Quality' session description are 'low', 'medium', and 'high'.

The potential combinations of session characteristics and the defined qualifiers could form an arbitrary large number of options. However, the EAT supports only the options that are defined within the Application Profile, and only these options are actually presented in the Regular Reservation GUI page.

The user has also to fill in two important sets of information about the application, which are the 'Connection Parameters' and the 'Service Schedule'. The Connection Parameters refer to the networking parameters of the application, such as the IP addresses and ports that are going to be used for the sessions (more details in the next section about the Advanced GUI). The EAT may know a priori these parameters (specified explicitly within the Application Profile) or may depend on the use of a Proxy (see section 2.5 for details). So, most probably, the user will not have to manually provide these parameters. Otherwise, the user has to be aware or able to find them out. The parameters of the Service Schedule are explained in [D1201] and are optional.

The user after selecting the desired options and filling in the necessary parameters submits the reservation to the EAT. The result of the reservation is presented in the user. In case of an unsuccessful reservation, the EAT notifies the user about the error, saying explicitly which is(are) the false parameter(s).

The GUI also contains some fields and selections that correspond to the Reservation Groups. These are described later in the Active Reservations GUI section.

Figure 2-7: The Advanced Reservation GUI

Figure 2-8: The Regular Reservation GUI

2.3.2.2 Advanced Reservation GUI

The Advanced Reservation GUI displays a full list of the reservation request parameters that have to be filled in directly by the "advanced" user (see Figure 2-7). The displayed form essentially presents all the contents of the reservation request message to be sent from the EAT to the ACA [D1201]. The construction of the Web page therefore directly depends on the AQUILA implementation (e.g. the specific parameters of the Traffic Spec).

The user has to fill in the following info:

- **Application identifiers**: the application identifiers are not used internally by the EAT, so they have only meaning for the user. They are used at the Release GUI to inform the user about the active reservations that can be released. The application identifiers are:

- Application Name: The indicative name of the application. The user can choose whatever name for the reservation; even this does not correspond to the real one.

- Service Component: The user has to select one of the choices in the provided list-box. This parameter is used to identify different components within the same application, again for illustration purposes only.

- **Network Service**: This part of the page is constructed dynamically according to the service offered by the network. The figure shows the currently defined network services, which are Premium CBR, Premium VBR, Premium Multimedia, and Premium Mission Critical. The user is allowed to select only one of them.

- The **Service Level Specification**: this is defined in [D1201] and consists of the following parameters:

  - **Scope** (Point-to-point, point-to-any, point-to-many, any-to-point). The selected Scope has to match with the selected Network Service (checked upon submission of the request).

  - **Flow identifiers**: These are the network connection parameters (similar to the Regular reservation case) and consist of the fields for: the 'IP Address', the 'Net-Mask', the 'Lower Port' and 'Upper Port' numbers for both the Source and Destination sides. The Lower Port is mandatory to be specified. If the user specifies also the Upper Port, the EAT will make the reservation for all ports greater than or equal to the lower port up to the upper port (included). Alternatively, the user can select one of the available Proxies (from the 'Proxy' list-box) to let the system detect these parameters. There is also a "Bi-directional reservation" check-box, which has the meaning that when ticked (selected), the EAT will perform the reservation in both directions from source to destination as well from destination to source (with the same reservation parameters).

  - **Traffic Specification**: It consists of the traffic conditioning parameters like Peak Rate (PR), Bucket Size for PR, Sustainable Rate (SR), Bucket Size for SR, Minimum policed unit, Maximum Packet size, Expected average rate, 1st threshold for Bilevel, 2nd threshold for Bilevel[3]. The units for each parameter are also presented, to facilitate the user to fill in the field with the correct value.

  - **Service Schedule**: the schedule has to do with the timing schedule for this reservation and is included in the GUI mainly to facilitate the placement of a reservation much before the actual service time. It consists of the 'Start Time', 'Duration', 'Idle' and 'Cycles' fields. These fields are only optionally filled-in, and in case they are empty, the reservation is accomplished with the notion 'from now on until explicitly released'.

[3] Some of these parameters are not really used within the EAT, but are left in the GUI for illustration purposes.

The user after filling in all the appropriate parameters can submit the reservation. The parameters are going to be validated at two levels (the EAT and the ACA) and if the reservation is successful the portal will indicate it to the user. In case of false validation, the portal will display the same Reservation page with the system-provided error message on the top of the page, as well as some specific error messages on top of each field, where the error(s) is(are) located. The previously entered parameters are not lost, but displayed again in order for the user to be able to correct or delete them, or add new ones according to the error message.

2.3.3 Active Reservations

The Active Reservations GUI provides the opportunity to the user to see and manipulate the active reservations. In a few words, the user can release reservations, or organise them in groups. The EAT supports the concept of Reservation Units and Reservation Groups. Reservation units refer to single reservations or to a set of reservations that will be treated as a single reservation (e.g. a bi-directional reservation). The reservation groups are used to collect a set of units or groups under one logical group.

The Active Reservations GUI presents in two parallel panels the active reservation units and groups. In the left panel, all the units and (some of the) groups[4] (i.e. their identifiers) are shown. In the right panel the internal units/groups of all reservation groups are presented. The GUI provides functionality for a user to:

- Release a reservation unit or a reservation group: The user can select one, more or all units/groups and release them. Releasing a group means that all aggregated members (i.e. the reservation elements of the units, and the units and groups of the groups, resp.) are released.

- Create, delete a group: The user can create a new group to later join units or other groups into it. The user can delete a group. By deleting the group, the aggregated members within units and groups are not released.

- Join reservation units or groups into a group: The user can select a unit or a group and join it into another group. A group can be a sub-group of only one group.

Having in mind the two reservation request modes of the portal (regular and advanced), the Regular Reservation mode can inherently support reservation units and groups, because the user selects the desired QoS for all sessions of this application. The Regular Reservation GUI contains a ‘Create Group’ check-box, which when selected means that the EAT will automatically create a group, join all these sessions into it, and submit the grouped reservation to the ACA. Otherwise the reservations are treated as individual reservations. In the Regular Reservation GUI there is also a “bi-directional reservation” check-box which will make both reservations from the source to the destination and from the destination to the source, with the same QoS parameters. In this case, the two reservations are considered as one reservation unit.

[4] It will be explained below which group identifiers are presented in the left panel.

Moreover, the Regular Reservation GUI contains a 'Complete Group Needed' check-box, which indicates that all reservations for the service components of the application have to be treated as a Reservation Unit.

In the Advanced Reservation mode, the scenario is somehow different, because the user is not able to specify all reservations at once, but one at a time. So, reservations created in the Advanced Reservation mode are always submitted as individual reservations (units). The user can however later create a new group and join any of those reservations into it. The Advanced Reservation GUI contains however a "bi-directional reservation" check-box, which has the same usage and effect as that of the Regular Reservation GUI.

The Active Reservations GUI will display in two panels the active reservation units and groups. The user is able to select a reservation unit or group from the left panel and join it to a selected group selected in the right panel, by pressing the 'Join' button. Furthermore, the user can select a group and release or delete it, by pressing the 'Delete' button. Finally, the user can press the 'Create' button to create a new group. In this case, the user has to enter a name for the group in the "Group Name" edit box.

As an example, an instance of the Active Reservation GUI could be that of Figure 2-9. We can see that the user has created two groups with names 'NetMeeting A' and 'NetMeeting B', which have been grouped under the 'NetMeeting' Group. 'NetMeeting A' and 'NetMeeting B' each has two reservations ('Audio a' and 'Video a', 'Audio b' and 'Video b', respectively). These reservation groups may correspond to the same meeting among three participants, and that's why these groups have been grouped into the 'NetMeeting' Group. Moreover, the group 'Download' contains two reservations, one for 'Music' download and one for a 'Video' download. Finally, there are two reservation units with the name 'VoIP' and 'FTP'. The 'VoIP' unit could be a bundle of two bi-directional reservations. In the left panel, also the 'NetMeeting' and 'Download' groups exist, because they can be joined in another group. 'NetMeeting A' and 'NetMeeting B' are already part of a group, and therefore they cannot be associated with some other group. That's why they are not listed in the left panel.

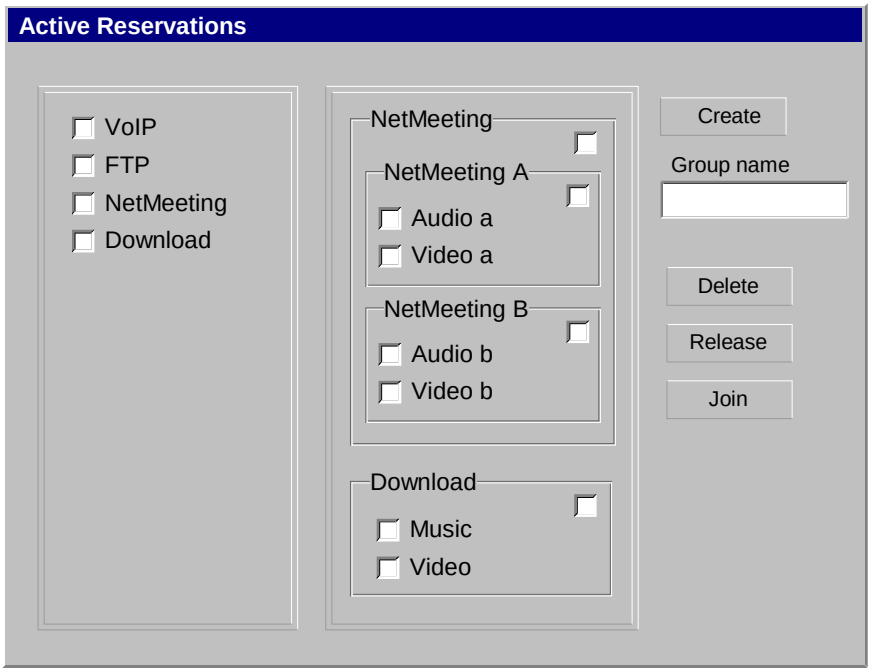

Figure 2-9: The Active Reservations GUI

The operations that can be performed having this instance of the Active Reservations GUI in mind are the following:

- Select the 'VoIP' or 'FTP' reservation unit and 'Join' it to any of the four groups.
- Select the 'Download' Group and 'Join' it to the 'NetMeeting' group (or vice versa). The 'NetMeeting A' and 'NetMeeting B' groups cannot be joined to any other group.
- Select any of the units or groups and 'Release' them. Releasing the 'NetMeeting' group would result in the release of the 'NetMeeting A' and 'NetMeeting B' groups, as well as the release of the contained reservations ('Audio a', 'Video a', 'Audio b', 'Video b').
- Select any of the groups and 'Delete' them. Deleting the 'NetMeeting ' group would result in the de-association of the 'NetMeeting A' and 'NetMeeting B' groups from it (they would not be deleted!). Subsequently, 'NetMeeting A' and 'NetMeeting B' will be listed in the left panel.
- Create a new (empty) Group, by filling in the 'Group Name' edit box with the wanted name for the new group, and then pressing 'Create'.

2.3.4 Logout

The user may perform the "logout" operation at any time. The Logout GUI contains a check box that asks the user whether he/she wants to release all reservations first. Then the user either confirms the logout operation, by pressing the 'Logout' button, or cancels the operation

by pressing the ‘Cancel’ button. If the user does not release the reservations, and later wants to perform some actions with them, he/she has to re-login.

The concept of logging out without releasing the active reservations can be understood in the following scenarios. Suppose that the reservation is placed from a third-person (an administrator) on behalf of the actual users for a videoconference. The third-person, after making the reservation, can logout from the portal, but still enable the participants to continue with the QoS provisioned session. Another scenario is when a user has made a reservation for a big file download. If the user wants to leave the PC for a while during the download, he/she can logout without releasing to make sure that no other user will release, accidentally or not, the reservation.

2.3.5 Navigation/Appearance

Although the design and development of the portal focus on the functionality and its correct operation, we also followed some basic rules that contribute to the navigability and appearance of the portal. Navigation is supported by the existence of two frames, one for the menu and the other for the actual GUIs. The upper frame contains the main menu, from which the user can navigate through the various GUIs, which open in the lower frame. For example, the main menu and the Regular Reservation GUI are as shown in Figure 2-10.

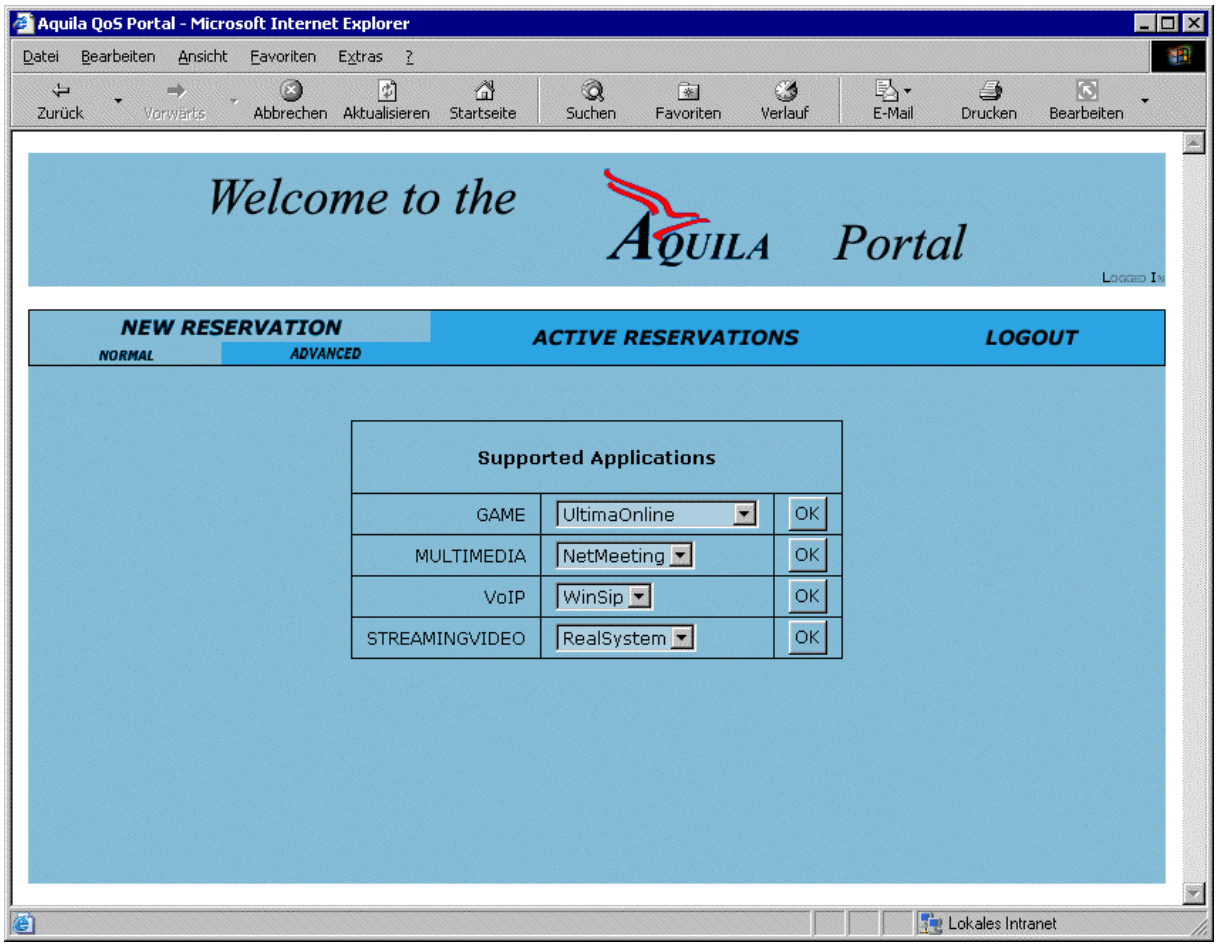

Figure 2-10 :The AQUILA Portal – the main menu and the Regular Reservation GUI[5]

2.4 Application Profiles

2.4.1 Definition Application Profiles

As application development is out of the scope of the AQUILA project, we need a mechanism to support existing legacy application. Legacy applications are not QoS aware. They are neither able to offer QoS to end-users nor to request for QoS on their own. The EAT and the converter provide mechanisms for offering and requesting QoS for legacy applications and therefore use the Application Profiles. Application Profiles are the mean to QoS enable legacy applications. With Application Profiles and their specifications it is possible in a first step, to offer QoS and to request for QoS from a QoS portal independently from the legacy applications. In the future it is conceivable to use Application Profiles to specify new QoS aware applications (where the QoS offer and request are integrated in the application itself).

[5] "Normal" in the picture means the Regular Reservation mode.

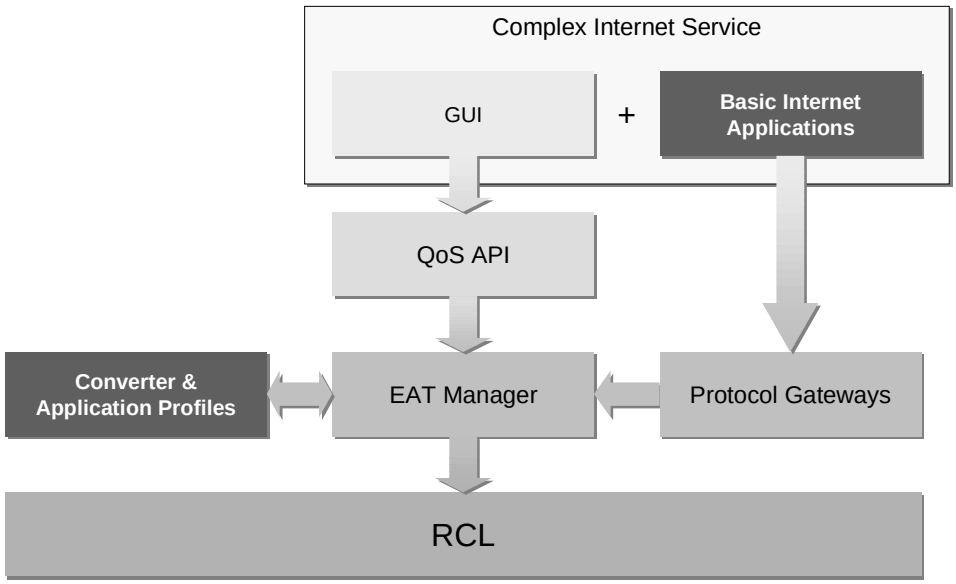

Figure 2-11 : Example of the architecture in the complex Internet service scenario showing the combination of non-QoS aware Basic Internet Applications and Application Profiles

The Application Profiles provide the whole necessary data about legacy application for these mechanisms to work properly.

The aim of the Application Profile is to specify as much as possible an application at network, application and end-user level. An Application Profile should provide a universal description of applications at these three levels independently from network operations. An Application Profile should provide a general description of the QoS request toward a network at application level (how does the produced traffic looks like – traffic specification, how are the QoS requirements – QoS Specification, what can an end-user sets up that have an implication on the QoS request – Session characteristic specification, NS Specification).

The converter is able to convert a neutral application QoS request based on the profile into a proprietary (AQUILA for example) QoS request. The EAT provides a neutral QoS API for Internet services or future application developers based on the Application Profiles.

Application Profiles provide:

- Specification at application level (in term of a DTD) of
  - End-user oriented information: *session characteristics*
  - *QoS requirements*
- Specification of legacy application at network level (in term of a DTD) of
  - Network and technical oriented information: *technical characteristics*

- *Codecs*

- Concrete legacy application information (in term of XML files)
  - Predefinition of *session characteristics* sets in order to keep the offer at end-user level consistent. Each application component is only described once.
  - Predefinition of *QoS requirement* sets. All applications of a same type have the same QoS requirements.
  - *Technical characteristic* sets

2.4.2 Specification of Application Profiles at Application Level

At application level it is possible to find common descriptors that can be specified. Concrete applications (for example NetMeeting from Microsoft) at high level can be assigned to an *application type* (for example video conferencing tool). As suggested by the *accountability knowledge level* pattern it is possible for the description of applications to differentiate between a *knowledge level* – the specification level, and an *operational level* – concrete application. An application type itself can be as well described at high level by so called *service component types* (for example video, voice, data etc.). Each service component type has common *QoS requirements* toward the network function of throughput, delay, jitter, loss and error that can be specified. In the same way a service component has common user friendly descriptions the *session characteristics*.

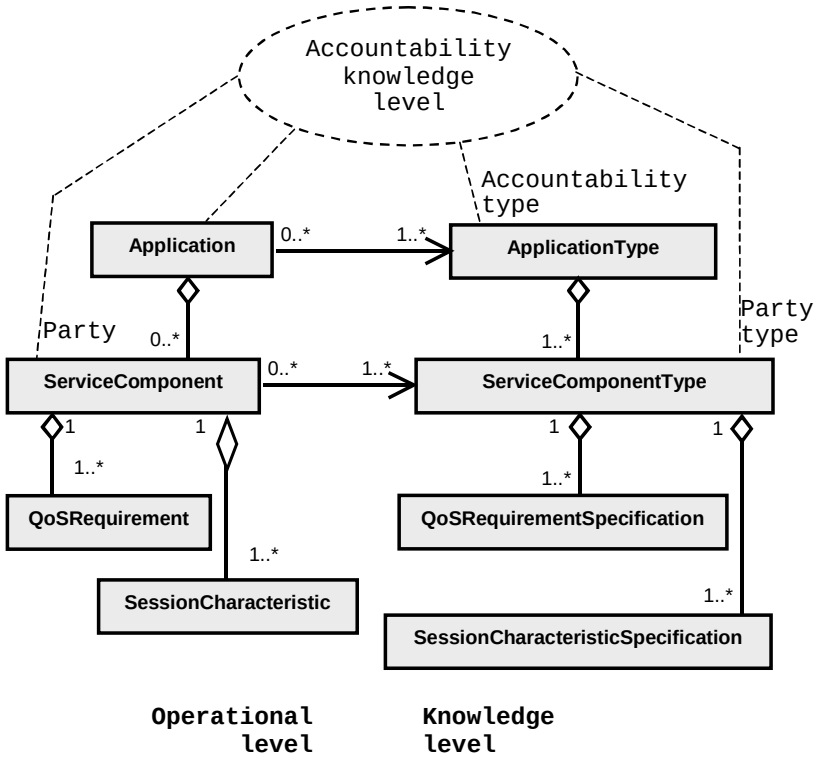

Figure 2-12: Application analysis diagram and the accountability knowledge level pattern

2.4.3 Specification of Application Profiles at Network Level

The description and specification of application features is at network level not a trivial task, and cannot be systematically achieved.

The following UML diagram opposes the systematic description of application (in white) to the unsystematic (in grey) one.

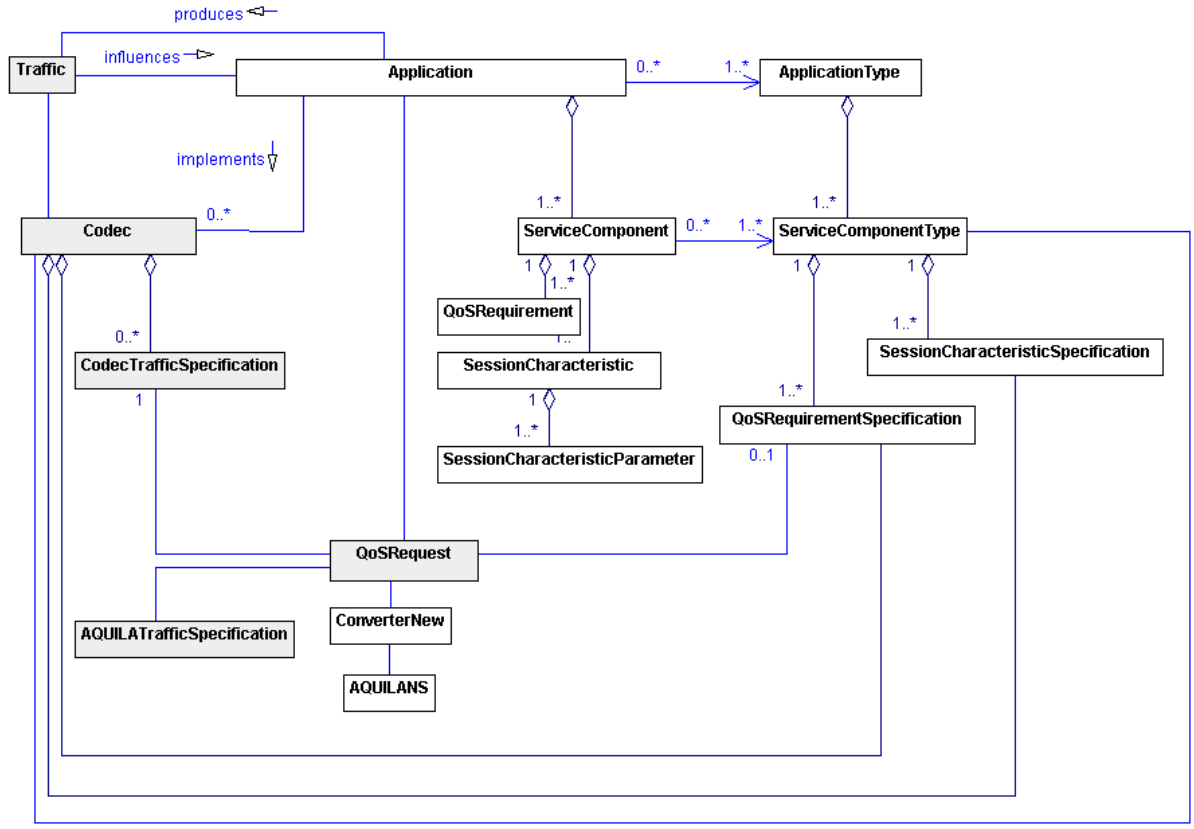

Figure 2-13: Analysis UML class diagram of the application at application and network level

The produced complex traffic flows do not depend on general application types but on the concrete implementation issues and strategies. The implementation of the codecs used is relevant for the description of the traffic produced. Table 2-2 depicts parameters at different levels (like traffic type, connection rate, living time, bit rate, micro-flows, and packet size) that can vary from stream traffic to elastic traffic, or from bursty traffic to constant traffic, and describe traffic flows.

Factor	Alternative 1	Alternative 2
Traffic type	Stream: time integrity preservation	Elastic: loose time requirements
Burstiness	Bursty	constant
Connection rate	High rate	Low rate
Living time	Long	Short
Bit rate	Constant bit rate	Variable bit rate
Microflows	Multiple	Single
Packet size	large	small

Table 2-2: Traffic descriptor and possible values

To be competitive and usable codecs have to cope with the actual best effort network, and respect as much as they can the QoS requirements (like delay / latency control, bandwidth

minimisation) of application types. Therefore they implement different mechanisms. For example, for bandwidth minimisation [Codecs] a solution in employing a variable bit rate transmission corresponding to a non-constant representation of the data (in the case of voice this can be silence suppression, or optimisation of the compression bit stream). These solutions are an obstacle when it comes to describe or specify the out coming traffic.

The Application Profiles concentrate in a first step on non-adaptive codecs.

The Application Profiles refer to codec Profiles, mainly because many different applications implement the same codecs.

2.4.4 Utilisation

The Application Profiles are specified with DTDs / XML schemas (see Appendix). In a first step the developer creating the Application Profile has to collect the information concerning the application itself, namely name, version, build, type, and scope. In a second step he/she has to collect the information concerning the implementation, namely: the codecs involved, the transport protocol, and the other protocols. If the codecs involved are not already analysed we recommend doing so and creating a profile. Concerning the implemented protocols the developer has to find out information concerning the port used (upper port number, lower port number, control port, etc.).

The developer has to provide the information marked in grey.

```
<!ELEMENT ApplicationProfile (CodecImplementation+, protocol*)>
<!ATTLIST ApplicationProfile
 name CDATA #REQUIRED
 version CDATA #REQUIRED
 build CDATA #IMPLIED
 type (VoIP | MULTIMEDIA | STREAMINGVIDEO | STREAMINGAUDIO | GAME |
 OTHER) #REQUIRED
 scope (unspecified | unidirectional | bidirectional | p2p |
 xdirectional) #REQUIRED
>

<!ELEMENT CodecImplementation (Codec,TransportProtocol)>
   <!ELEMENT Codec (#PCDATA)>
   <!ATTLIST Codec
    xlink:form CDATA #FIXED "simple"
    href CDATA #REQUIRED
   >
   <!ELEMENT TransportProtocol EMPTY>
   <!ATTLIST TransportProtocol
                name (TCP | UDP) "TCP"
   >
<!ELEMENT protocol (lowerPortNo?,upperPortNo?,isControlPort?)>
<!ATTLIST protocol
 name (unspecified | RTP | RTSP | RSVP | SIP | SDP | H320 | H321 |
 H322 | H323 | H324) "unspecified"
>
   <!ELEMENT lowerPortNo (#PCDATA)>
   <!ATTLIST lowerPortNo
    value (unspecified | fixed | configurable) "fixed"
   >
   <!ELEMENT upperPortNo (#PCDATA)>
   <!ATTLIST upperPortNo
    value (unspecified | fixed | configurable) "fixed"
```

```
   >
   <!ELEMENT isControlPort (#PCDATA)>
   <!ATTLIST isControlPort
    value (unspecified | yes | no) "no"
   >
<!ELEMENT description (#PCDATA)>



<!ELEMENT ApplicationProfile (CodecImplementation+, protocol*)>
<!ATTLIST ApplicationProfile
 name CDATA #REQUIRED
 version CDATA #REQUIRED
 build CDATA #IMPLIED
 type (VoIP | MULTIMEDIA | STREAMINGVIDEO | STREAMINGAUDIO | GAME | OTHER) #REQUIRED
 scope (unidirectional | bidirectional | p2p | xdirectional) #REQUIRED
>
```

Concerning the preparation of the Codec Profiles. Some parts are predefined and prepared (like session characteristics, QoS requirements, codecs), other have to be manually investigated and filled in (like traffic information) carefully.

2.5 Proxies

Applications, especially multimedia ones, usually open data connections with the use of connection setup protocols. Example protocols are the Session Initiation Protocol (SIP) [SIP] and the H.323 protocol suite [H.323] for the establishment of IP telephony calls. For the Multi-Field (MF) classification and marking of those user flows at the edge router, the source and destination IP addresses and TCP/UDP port numbers of the connection are required. However, the port numbers are always not well-known and are usually negotiated during the call setup.

2.5.1 Use of the Proxies

The main goal of the Proxies or Protocol Gateways is to interfere in the connection setup protocol exchange. In this way, they are able to intercept and translate the exchanged messages. They can extract their content, find the required parameters and to pass them to the EAT.

The Proxy Framework of the EAT provides a set of Application-level Proxies or Protocol Gateways. There is one Proxy for each popular connection setup protocol. The Proxies are administered by an entity called the Proxy Manager. The Proxy Manager has the following roles: To locate the available Proxies and to mediate between them and the EAT.

There are two modes of reservation requests and respectively, two modes of Proxy operation.

2.5.1.1 GUI-initiated Reservation

In the first case, the user initiates a reservation request through one of the GUIs (Advanced or Usual Reservation GUI). In the case that the port numbers of the legacy application are not known beforehand and a Proxy is available for this application, the Proxy Manager is con-

tacted by the EAT. The Proxy Manager should then find the appropriate Proxy, communicate with it (with CORBA) and request for the port numbers.

When this information is retrieved by the Proxy, it asynchronously notifies the Proxy Manager, who in turn notifies the EAT. The Proxy may forward as much information as it can retrieve, such as information about the required bandwidth or the codecs used for the session. The GUI-initiated approach was used during the 1st trial.

2.5.1.2 Proxy-initiated Reservation

On the Proxy-initiated reservation, the Proxy detects the start of a new connection by inspecting the messages that pass through it. It then initiates the resource reservation request by notifying the Proxy Manager. This notification contains all the information that the specific Proxy can supply and consists of some or all of the following parameters:

- source and destination IP addresses
- destination TCP/UDP port number (also the source port number, if available)
- the type of the service component (audio, video)
- codec (if one is used for media encoding)
- expected bandwidth
- name or SIP address of the requester

The EAT Manager decides whether to continue with this resource request or not, depending on whether the supplied information is sufficient in terms of user authentication and requested resources. This means that the information furnished by the Proxy is sufficient for the calculation of the reservation request parameters by the Converter module. It also means that the requesting user should be registered in the EAT and that his/her preferences allow for automatic reservation setup by the EAT. A user may prefer that all her SIP calls be automatically accepted by the EAT and that a reservation request be sent to the RCL. Another one may define a limited set of destinations for SIP calls that should be accepted, while the rest should be (silently) not given QoS.

2.5.2 Implementation of a new Proxy

Although the Proxies are part of the End-User Application Toolkit, they do not run in the same process as the rest of the EAT, but in their own processes. This is due to the distributed nature of the EAT and the need for the Proxies to be flexible and located in various sites (for example a Packet Filter should be placed in the border of the access network). Therefore, a new Proxy must be implemented as an independent program.

The Proxies communicate with the Proxy Manager through a well-defined CORBA interface (ProxyManager interface). A new Proxy can be created (or an existing Proxy may be modified) to use the *ProxyManager* interface in order to signal to the Proxy Manager that a new connection was detected and to provide the corresponding parameters.

```
/*
    Package aquila.rcl.eat.proxy
    File    proxy.idl
    Version 3.7, 11.12.2001

    National Technical University of Athens (NTU)
    Dresden University of Technology (TUD)

    This file contains the interface of the Application Proxies

    IMPORTANT: The ProxyManager within the proxy package also uses/implements
    entities defined in eatManager.idl.
*/

#ifndef proxy_idl
#define proxy_idl

#include "aca.idl"
#include "eatManager.idl"

module proxy {

    struct ProxyDescription {
        long   proxyId;
        string proxyName;
        string proxyDescription;
        long   controlPort;
    };

    typedef sequence <ProxyDescription> ProxyDescriptionSeq;

    exception ProxyException {
        string reason;
    };

    interface ApplicationProxy {

        readonly attribute ProxyDescription proxyDescr;

        void registerApplication (
            in long         sessionId,
            in long         proxyId,
            in string       sCompName,
            in aca::Address sourceAddress,
            in aca::Address destinationAddress,
            in boolean      isBidirectional)
            raises (ProxyException);
    };

    interface ProxyManager : ApplicationProxy, eatManager::SessionStarter {

        const string SERVER_NAME = "ProxyManager";
        const string SERVER_KIND = "Proxy manager object";

        /**
         * To register an app proxy at the Proxy Manager.
         */
        void registerProxy (in ApplicationProxy appProxy);

        /**
```

```
         * To provide the information only for the EAT Manager and the GUIs.
         */
        ProxyDescriptionSeq getApplicationProxies ();
    };
};

#endif // proxy_idl
```

When a new connection is detected, the establishQoS( ) function is called, notifying the Proxy Manager. The Proxy should wait for the outcome of the reservation process (success or failure) that is given by the function return code. In the same way, a reservation may be released if the Proxy detects that the call was released too.

So far, the Proxy-initiated reservation scenario has been dealt with. A Proxy however, should also be prepared to answer questions from the EAT about a reservation (GUI-initiated scenario). For this purpose, it must implement the *ApplicationProxy* interface, also presented in the previous listing. The Proxy Manager calls the registerApplication( ) function in order to ask the specific Proxy for the port numbers of a connection, providing its IP addresses (source and destination). When this information is detected, the Proxy will reply asynchronously, using again the establishQoS( ) call of the ProxyManager interface.

Please note that the ProxyManager interface is an extension of two interfaces: ApplicationProxy and SessionStarter. The latter is presented in the following listing.

```
/*
    Package aquila.rcl.eat.eatManager
    File    eatManager.idl
    Version 3.6, 11.12.2001

    Dresden University of Technology (TUD)
    National Technical University of Athens (NTU)

    This file contains the specification of the internal interface of the EAT
    Manager towards the Proxy, and the Converter.
*/

#ifndef eatManager_idl
#define eatManager_idl

#include "aca.idl"
#include "service.idl"

module eatManager {

    /**
     * ManagerException containing a message string.
     */
    exception ManagerException {
        string reason;
    };

    /**
     * The session information consits of data detected by the Proxy in order to
     * allow a correct reservation request.
     */
    struct Session {
        long            sessionId;
        long            proxyId;
        aca::Address    sourceAddress;
        aca::Address    destinationAddress;
```

```
        aca::ProtocolID protocol;
        string          serviceComponent;   // audio or video
        string          payloadType;
        long            payloadNumber;
        long            bandwidth;
        string          requesterName;
    };


    /* --- Session Starter ------------------------------------------------- */

    /**
     * The SessionStarter interface is implemented by the EAT in order to be
     * informed when the network/session information for an application is
     * available and the reservation request at the ACA can be startet.
     */
    interface SessionStarter {

        const string SERVER_NAME = "SessionStarter";
        const string SERVER_KIND = "EAT Manager session starter object";

        /**
         * A proxy or the ProxyManager calls this function to send/return the
         * session information to the Proxy/EAT Manager in order to allow a
         * reservation request.
         * 'Send' means that the request is initiated by the Proxy itself.
         * 'Return' means that the request is initiated by the end-user, but
         * without knowledge of the session details. So the Proxy helps by
         * responding the registerApplication() message from the EAT Manager.
         *
         * @param  sessionId      The Id of the session. In the case of an user
         *                        initiated request, the id is choosen by the
         *                        EATManager as index. In the case of an proxy
         *                        initiated request it is -1. The EAT Manager
         *                        then returns the newly created session id.
         * @param  sessionInfo    The identified session information for this
         *                        Session.
         * @param  isBirectional  Indicates whether a reservation has to be
         *                        requested for both directions or not.
         * @return                Is true if the reservation has been
         *                        successfully established.
         */
        boolean establishQoS (
            inout long sessionId,
            in Session sessionInfo,
            in boolean isBidirectional)
            raises (ManagerException);

        void releaseQoS (
            in long sessionId)
            raises (ManagerException);
    };
};

#endif // eatManager_idl
```

During startup, the Proxy should register with the Proxy Manager, provide a reference to itself as well as its description to the PM, using the registerProxy( ) call.

2.6 EAT Script

The following chapter contains a short description of the “EAT Script”, useful for a batch processing of automatic logins, reservation requests, reservation releases, and logouts. A

command line tool has been realised that is able to parse the script and to perform the related actions at the EAT.

The aim of such a mechanism is to support automatic test scenarios. Also the measurement people may use it for their purposes.

2.6.1 Script Specification

XML is used for the specification of the script due to the following reasons:

- XML is a well established, flexible, easy to understand language for structuring data and documents [XML, Harold98, Jelliffe98].
- There are some free tools existing for the creation and modification of XML files:
  - XML and Web Services DE: http://www.alphaworks.ibm.com/tech/wsde
  - XML Notepad: http://msdn.microsoft.com/xml/notepad/intro.asp
  - Visual XML: http://www.pierlou.com/visxml/
  - xmloperator: http://www.xmloperator.org/
  - See also: http://www.garshol.priv.no/download/xmltools/
- Java APIs exist for the parsing of XML documents, e.g. [JAXP].
- The Application Profiles are also based on XML.

2.6.1.1 Document Type Definition

The DTD (EATScript.dtd, see Appendix) specifies the permissible elements and their structure within each EAT script. It has to be installed in the same directory as the XML scripts themselves. It is not useful to modify it but only to use it for syntax and semantic checks.

The root element is called `EATScript`. The script can optionally start with an `Import` element in order to "import" identifiers (`LoginName`, `RequestId`) of previous scripts. But generally, the `EATScript` consists of `Login`, `Request`, `Release`, and `Logout` elements. Their order and the number is free in terms of the syntax. (However, it makes no sense to release a reservation before requesting it. Note also that a logout also releases all reservations of the belonging user.)

A `Login` requires two attributes: an identifier `LoginName` and a `Password`. `Request` and `Logout` elements have to refer to the above `LoginName`.

A `Request` requires its own identifier `RequestId` that is referred by `Release`. Moreover, `Request` consists of one or more `RequestSpec` elements that require a list of elements

(`NetworkService`, `SLS`) that is related to the actual advanced request interface as it is specified in api.idl. If more than one `RequestSpec` elements are defined in a `Request`, a multiple reservation unit is automatically built. However, reservation groups are not supported.

For the `TrafficSpec` elements, the units of the values are given as attributes. However, these units are at the moment fixed, because they mainly support the script editors by specifying the values.

2.6.1.2 XML Examples

In the Appendix, two sample XML files are given: The first script (Example.xml) includes one login, two requests, and the release of the first request. The requests are for two different network services and are filled by some "reasonable" values. The second request consists of two reservation (`RequestSpec`) elements that form a bi-directional reservation. The second script (Example1.xml) then contains the release for the second of the above mentioned reservations.

2.6.2 Command Line Tool

A command line tool has been realised that allows the parsing and execution of an XML file's content. This tool is implemented by the Java class EATScript.java. To parse an EAT script file[6] type:

```
java aquila.rcl.eat.script.EATScript [ORBOptions] [-v] xml-file
```

(The optional parameter –v switches the verbose mode on so that additional parsing messages are printed.)

For example:

```
java aquila.rcl.eat.script.EATScript Example.xml
```

The EAT script parser is based on the Sun's Java API for XML Parsing [JAXP], version 1.1. Download this API, install it, and add jaxp.jar as well as crimson.jar and xalan.jar to the CLASSPATH.

The parser communicates with the EAT via CORBA; the EAT must be running when the script starts.

EATScript.java itself is stateless, but the EAT Manager keeps all information about active end-users and reservations. So it is possible to refer to previous executed scripts by using the import elements. For example, a request that has been executed by the first script can be released by the second one:

```
java aquila.rcl.eat.script.EATScript Example1.xml
```

[6] Note that EATScript.java uses a validating parser. Therefore, the XML script should fulfil the DTD grammar.

3 Application Scenarios

3.1 Basic Internet Application Support

This section discusses about the way the EAT supports Basic Internet Applications. The latter are often legacy applications that cannot directly use the EAT (the EAT-provided API). We can think of the following classification of Basic Internet Applications, according to their QoS awareness:

- QoS-unaware applications. Such applications have no means to express their QoS requirements. The user is responsible to set-up reservations for such applications in two modes: advanced and regular. So, the support offered by the EAT is *user-initiated*.

- QoS-aware applications. Such applications can express their QoS requirements through signalling protocols, like SIP and RSVP, but they cannot be modified to use the EAT API. These applications are supported transparently for the user through the internal EAT modules and the use of the concept of User Profiles. So the support offered by the EAT is *EAT-initiated*. However, the user may select (by defining the User Profile appropriately) to not let the EAT make the reservations transparently. So, even for such applications the user-initiated approach may be used.

Apart from the QoS-awareness, the Basic Internet Applications can have single or multiple sessions. In the former case, the user or the EAT has to make one reservation for the single session of the application, while in the latter case the user or the EAT may perform more than one reservation, one for each session. The EAT therefore introduces the concept of the *reservation units* and *reservation groups*.

Reservation Units refer to single reservations or even multiple reservations that are treated by the EAT like a single one. Reservation units cannot be separated after creation and they can be deleted all at once. As an example, consider an IP telephony (VoIP) application. Usually, for such applications, a bi-directional reservation has to be established, both from the source to the destination and back from the destination to the source. These two reservations will form a single reservation unit.

Reservation Groups refer to multiple reservations that are logically gathered under one group. Groups are structures that give the possibility to the user to treat a set of reservations together. The difference from the reservations units is that in the groups the individual reservations can be de-associated from the group or released one by one. Considering a multimedia application, with one audio and one video session, the reservation group concept provides the user with the capability to treat these reservations as one logical entity. This facilitates the reservation and release of the grouped reservations in one movement, and adds user-friendliness to the approach; it is easier than having to reserve and release each one separately. However, it also provides the possibility to perform for example the release in one movement.

3.1.1 User-initiated Approach

The user can take advantage of the EAT capabilities through the use of the AQUILA Portal (see section 2.3). In order to start using the portal, the user must first login, providing his/her username and password. After successful authentication, the user can mainly perform reservations, release of active reservations, manipulation of the reservation groups, and finally to logout.

There are two main modes of reservation requests: the advanced mode and the regular one. The main difference is that for the former the user is supposed to know the general Internet QoS concept, some more specific AQUILA issues, as well as the exact QoS requirements of the application that is going to be used. On the other hand, the usual mode implies that all this knowledge is inherently provided by the EAT through the use of some technical developments like the Application Profiles and the Converter. In both cases, the user after specifying the required QoS (by filling in the actual reservation parameters or by selecting the desired prepared options respectively) will have to manually tune the application options, in order to be consistent with the requested QoS.

Independently of the reservation mode, the EAT may need to make use of the Proxy Framework to find out network connection parameters. In a few words, in order for the Proxy Framework to be able to identify these parameters, the application must be already launched and its sessions should be established. This fact implies that the reservations will actually be performed only after the user has established all application's sessions. Anyway, the duration of an active reservation (incl. accounting, charging, etc.) is always determined by the EAT and not by the application.

The release of the active reservations can be performed independently of the continuation of the application. The release can be performed either by explicitly using the Release GUI or implicitly by performing a logout.

3.1.2 EAT-initiated Support

The EAT takes advantage of the Proxy Framework to make reservations and releases on behalf of the user, based on the User Profile. The Proxy Framework is actually responsible for detecting new QoS-aware sessions and providing the EAT with all the necessary information to actually perform the reservations or releases. For this purpose, internal intelligent mechanisms are used to map the QoS requirements expressed in some formation (e.g. in RSVP Flowspec, or SDP session info) to the AQUILA-based reservation formation, within the Converter.

The users are however able to see the EAT-initiated active reservations through the normal use of the portal. Again, after a successful login, they can go to the Active Reservations GUI page and have a look at the performed reservations or choose to release them.

3.1.3 Reservation Units and Groups

The support of reservation units and groups provide the user with the following functionality:

- Make two or more reservations to be treated as one reservation unit, implying always simultaneous establishment or release of the reservation unit.
- Group together session's reservations that *physically* belong to the same application
- Group together session's reservations that *logically* belong to the same type of applications. This also implies that groups of reservation may be grouped together to form a higher-level group.
- Reservations belonging to the same group can then be treated as a unit, but also as individual reservations.

The existence of reservation groups provides easier manipulation of a bundle of reservations that physically or logically belong to the same group. Moreover it provides a more efficient support towards applications. Taking as an example an application with two (or more) reservations, the user is able to specify all of them and submit them as a group for the actual reservation to take place. This ensures that the reservations will be performed almost at the same time, preventing the user from being charged for a reservation for one session before the rest reservations are actually accomplished.

(More details about the actual use of reservation units and groups can be found in Section 2.3)

3.2 Complex Internet Service Support

A Complex Internet Service (CIS) consists of several Basic Internet Applications services like email, chat, video/audio streaming etc. The quality requirements on the Basic Internet Applications are different from user to user, depending on their technical knowledge, on their know-how of broadband media services and on their subjective impressions. To guarantee these very individual requirements, a corresponding QoS support must be set-up: The CIS may use the interfaces of the EAT (e.g. the API and the Application Profiles) to provide user-friendly QoS options at different levels towards its users.

First of all, a lot of technical preferences have to be selected in order to create a Complex Internet Service. These are for example the choice of the Basic Internet Applications, the interaction of the required components and the specifications of layout (e.g. screen size of a video). They have a direct or an indirect influence on the possible QoS parameters (e.g. the screen size of a video defines how much data has to be transferred in order to show the video in the selected quality).

The following sub chapters make recommendations for the QoS support of a Complex Internet Service. AQUILA Mediazine will be used as an example of CIS.

3.2.1 Recommendations for the Creation of QoS Level

Creating three categories of Internet connections is appropriate for Mediazine. This is a manageable number for the user, if he/she wants to specify his/her Internet connection in a more detailed way and it mirrors the up to date technical possibilities. Of course there are other possibilities of classification. The consequences of a classification modification are stated in chapter 3.2.3.

The three categories to classify today's existing Internet connections are:

1.	Low bandwidth connection	up to 128 kBit	e.g. modem/ISDN connection
2.	Medium bandwidth connection	128 kBit-2 MBit	e.g. DSL/cable modem connection
3.	High bandwidth connection	more than 2 MBit	e.g. LAN connection

The kind of connection influences quality features which are dependent on the limitation of the amount of transferred data per time interval, e.g. the size of the pictures of a video stream at a certain compression. To each connection category corresponds a category of services (QoS category), where a specific quality can be guaranteed:

Bronze
- time critical applications with a small amount of transferred data, e.g. action games
- services with speech input and output, e.g. Internet telephony
- audio streams, e.g. Internet radio

Silver
- same applications as under Bronze
- video/live streams with small or medium size of the pictures, e.g. Internet TV by a resolution of 320x240 pixels

Gold
- same applications as under Silver
- video/live streams in TV quality (720x576 pixels)

The exact allocation of the Basic Internet Application to one of the above mentioned categories should result from the technical features in combination with the respective Application Profiles of the services (see also chapter 2.4). The correlation between the QoS categories and the Application Profiles will be described in chapter 3.2.3.

In each category, the user encounters the following restrictions, depending on the bandwidth of his/her connection:

- At a low bandwidth, only Bronze QoS support can be guaranteed.
- At a medium bandwidth, the user can choose the warrantee of the Bronze or Silver category.
- At a high bandwidth, the user has free choice of all categories.

The user should be clearly informed that the higher the category is, the higher the costs will be. Indeed, the costs increase with the growth of offered and guaranteed bandwidth. The user will have to bear the additional costs, either directly (higher fee) or indirectly (more advertisement, release of personal information, etc.). The nomenclature of the categories was selected to make them easily recognisable and to help the user to create relations between them and the emerging costs.

3.2.2 Recommendations for the Creation of a Set-up Menu for the QoS Service

The users of Internet services can be divided into two groups according to their knowledge and experience in the use of computers and the Internet:

A. Technically experienced user

B. Novice

Both users correspond to "regular" user, which is described in chapter 2.3. As far as the example of a Complex Internet Service – AQUILA Mediazine – is concerned, the technical experienced user will be referred to as "Mediazine expert" and the novice will be referred to as "Mediazine novice".

The Mediazine expert can be asked for detailed information and technical terms such as the kind of connection, the bandwidth, etc. He/she can also deal with the corresponding information like latency, packet loss, etc. On the other hand, for the inexperienced user, such information should be described in general terms like "fast", "big", "CD quality", etc.

The afore descriptions result in particular specification criteria for the QoS support of a Complex Internet Service necessary for a set-up menu aiming at helping the user configure QoS based applications. The CIS set-up menu should start by testing the user's knowledge of computers and Internet (example see Figure 3-1). You will find notes to the following figures at the end of this sub chapter.

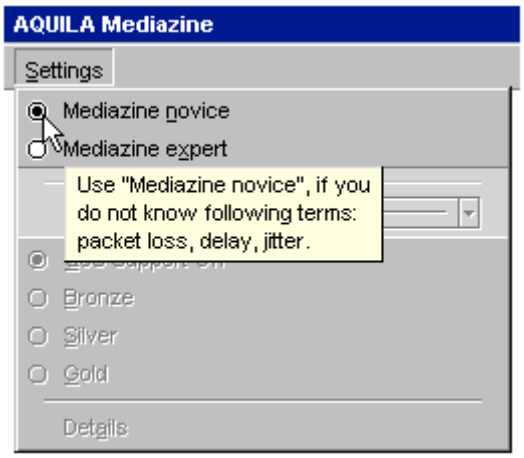

Figure 3-1: Settings: user selection

Thereafter, the following proceedings can be recommended:

a) Ask about the connection:

Mediazine expert	Mediazine novice
On the inquiry on the connection, additional details (28.8 kBit modem, 10 MBit LAN) can be asked for, in order to generate a better profile for the QoS support.	It is recommended to detect the kind of connection automatically. Should it not be possible, it is advisable to use general descriptive terms like e.g.: - slow connection (modem/ISDN) - medium fast connection (DSL/cable modem) - fast connection (LAN)

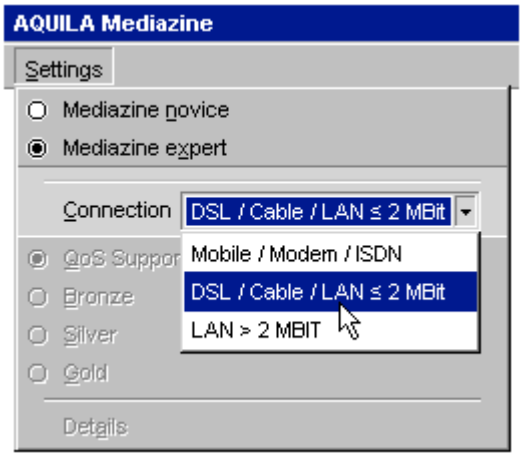

Figure 3-2: Settings: connection expert

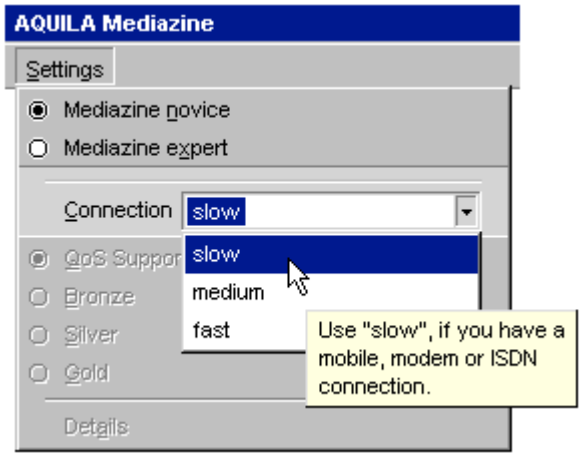

Figure 3-3: Settings: connection novice

Table 3-1: Mediazine connection settings

b) Ask about the QoS category:

Mediazine expert	Mediazine novice
The user has the possibility to adjust the QoS support settings, based on the present profiles of the various Basic Internet Applications or to use the standard user settings (see right side).	Depending on the connection, different levels of QoS support should be available, for example: - QoS support on/off - Bronze on/off by low bandwidth

- Bronze/Silver on/off by medium bandwidth
- Bronze/Silver/Gold on/off by high bandwidth

Depending on the selected category, the QoS settings should be adjusted with the help of the Basic Internet Applications profiles.

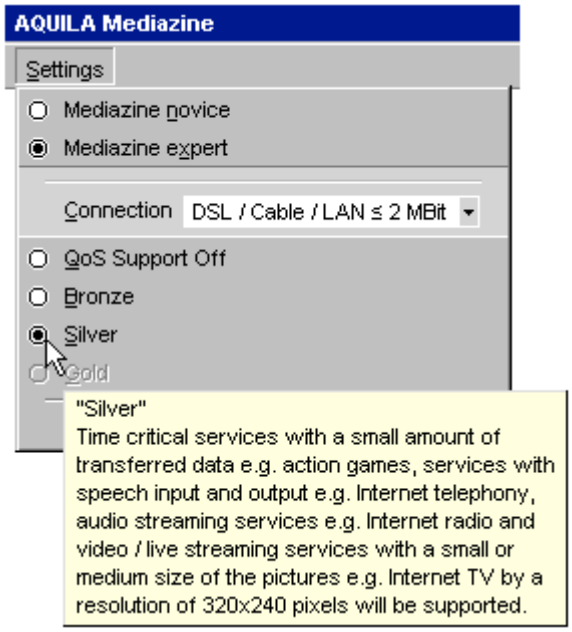
AQUILA Mediazine
Settings
Mediazine novice
Mediazine expert
Connection
DSL / Cable / LAN ≤ 2 MBit
QoS Support Off
Bronze
Silver
Gold
"Silver"
Time critical services with a small amount of transferred data e.g. action games, services with speech input and output e.g. Internet telephony, audio streaming services e.g. Internet radio and video / live streaming services with a small or medium size of the pictures e.g. Internet TV by a resolution of 320x240 pixels will be supported.

Figure 3-4: Settings: QoS category expert

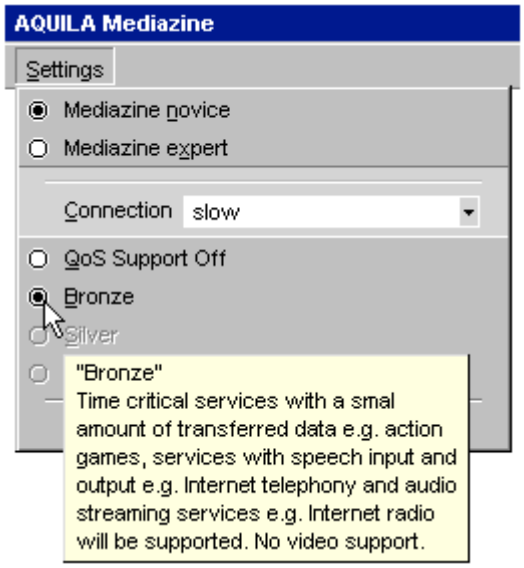
AQUILA Mediazine
Settings
Mediazine novice
Mediazine expert
Connection
slow
QoS Support Off
Bronze
Silver
"Bronze"
Time critical services with a smal amount of transferred data e.g. action games, services with speech input and output e.g. Internet telephony and audio streaming services e.g. Internet radio will be supported. No video support.

Figure 3-5: Settings: QoS category novice

Table 3-2: Mediazine QoS category settings

The relationship between the Complex Internet Service and the EAT (i.e. the Application Profiles) regarding the choice of the different QoS categories will be explained in a more detailed way in chapter 3.2.3.

For a better understanding of the selection possibilities in the set-up menu a help menu with extensive commentary should be available in addition to the “MouseOver” functions shown in the above figures. Especially the selection of the QoS category by the standard user requires further information. This will enhance the usability of the Complex Internet Service.

Furthermore it is recommended to allow the user to change the settings of the QoS service and his/her Internet connection on his/her own. This enables him to define his/her standard settings but also to have fast and uncomplicated access to special settings for special services.

One of these services could be an Internet-Live-Concert, that he/she wishes to see in category „Gold“ whilst his/her normal category is only „Bronze“.

3.2.3 Recommendations for the Interaction of the CIS and the EAT

During the phase of conception and development of a Complex Internet Service, basic conditions will have to be made, e.g. functions of the service, used Basic Internet Applications, layout etc. These conditions will influence the parameters that are necessary for the QoS support, e.g. the choice of the QoS category. In this context the following questions have to be answered: Which and how many categories are needed? In which way can a surplus value for the service provider be created by the usage of these categories? And in which way can this surplus value be quantified?

It is not the subject of this report to deal with billing systems or accounting systems or with questions of security and warranty that always occur in the usage of such a system. The business models will be dealt with in AQUILA document D3302. But the above mentioned questions are crucial for the assignment of the Basic Internet Applications and their relationship to the QoS categories (for further information see chapter 3.2.1).

The Basic Internet Applications can only be assigned by the development of a Complex Internet Service, because the conditions can only be defined there. It is recommended to use appropriate correlation profiles to allow better usage and easy modification. These profiles will be referred to as “QoS Category Profiles”.

Example of a “QoS Category Profile” for AQUILA Mediazine:

Example of QoS Profile

QoS Category : Bronze

Application Name	Service Component	Options	Default Option
RealPlayer	AUDIO	Option 2	Option 2
		Option 3	
	VIDEO		No Support
NetMeeting	AUDIO	Option 2	Option 2
		Option 3	
	VIDEO		No Support
Ultima Online	GAME	Option 1	Option 1
		Option 2	

Figure 3-6: QoS Profile Mediazine

This profile describes the correlation between a special QoS category and the options of a Basic Internet Application Profile by using the “Application Name” and the “Service Compo-

nent". For each Basic Internet Application and its service components, one of the options will be used as default option (see also Figure 3-6).

The Mediazine expert has to have the possibility to change the QoS parameters within the CIS set-up menu in detail.

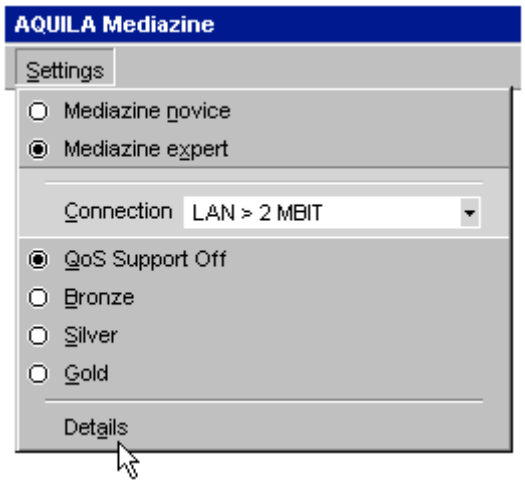

Figure 3-7: Settings: all QoS cat. exp. user

Example: Typical values of such QoS parameters used for the service WinAmp, an application to play audio and video files.

ApplicationName = WinAmp

Version = 2.77

Location = C:\Programme\Winamp\Winamp.exe

SupportedMediaTypes = mp3, mp2, wav

QoS Category = Bronze

Quality = medium quality

Bandwidth [KB/s] = 64

Sampling rate [KHz] = 44

QoS Category = Silver

Quality = CD quality

Bandwidth [KB/s] = 256

Sampling rate [KHz] = 44

As can easily be seen, a part of the data depends on the characteristics of the service used by WinAmp or of WinAmp itself, e.g. the programme version, the location of the application and the assignment of the categories. This data should be available within the Complex Internet Service. Other parts are QoS specific, containing the bandwidth and quality of the transmission. This data can be set in the profiles of the Basic Internet Applications, as they are provided and used by the EAT.

In case of the EAT the necessary QoS parameters of the Basic Internet Applications can be extracted from the respective Application Profiles (see also chapter 2.4 and Figure 3-8). For it the EAT API offers an interface named `ApplicationManager` with the function `getAvailableApps():ApplicationInformation[]` (look at chapter 2.2). To explore the usage of the EAT API for the creation of a QoS session see chapter 2.2 (there you will find a usage scenario in section 2.2.2).

Example of Application Profile

Application Name : RealPlayer
Application Type : MULTIMEDIA

Service Component : AUDIO

	Audio Quality	Transfer Rate	Sampling Rate
Option 1	"CD quality"	256 kBit	44 kHz
Option 2	"low quality"	64 kBit	44 kHz
Option 3	No Reservation (Best Effort)		

Service Component : VIDEO

	Video Quality	Window Size	X [pixel]	Y [pixel]
Option 1	"TV quality"	"large"	720	576
Option 2	"medium quality"	"medium"	320	240
Option 3	"low quality"	"small"	160	120
Option 4	No Reservation (Best Effort)			

Figure 3-8: Application Profile RealPlayer

The relationships between the QoS profiles and the Application Profiles are defined by the parameter “Option” with the “OptionID”. For the description of this parameter see also the Codec Specification DTD (Codec Profile) in the Appendix. Figure 3-9 illustrates a relationship between the Mediazine QoS profile and an EAT Application Profile for the Basic Internet Application RealPlayer.

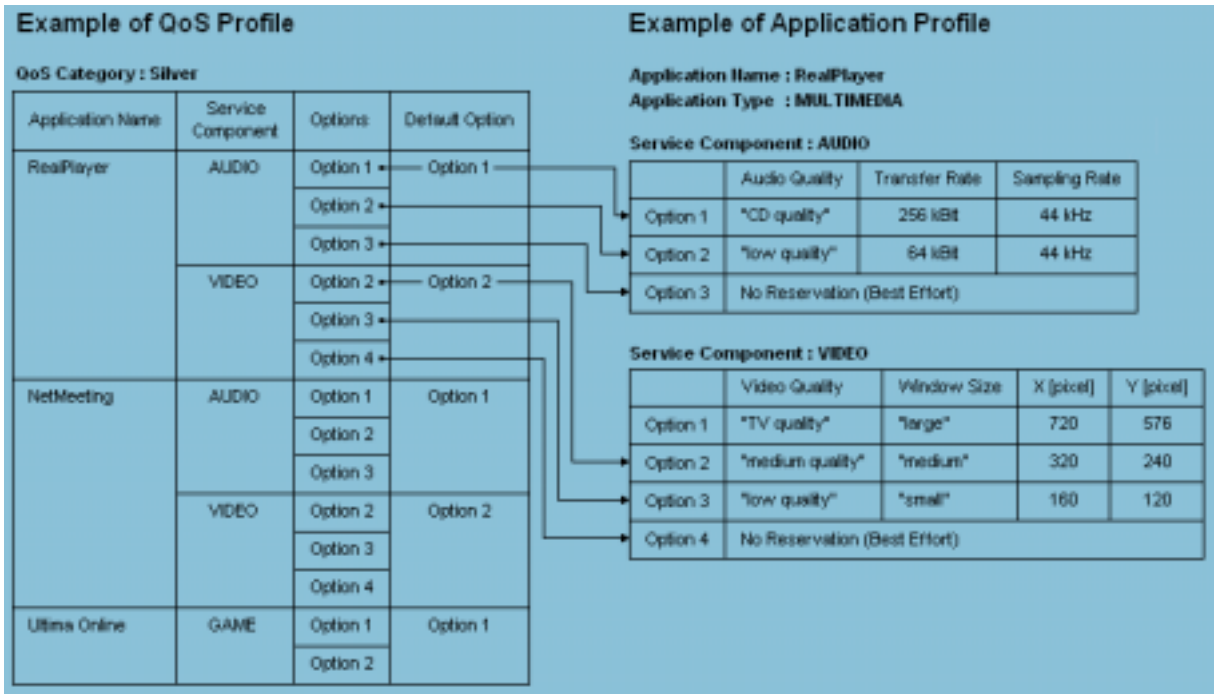

Figure 3-9: Correlation between QoS Profile and Application Profile

It is not possible to define exactly which data will have to be saved in which way and in which profile. Whether for the development of a Complex Internet Service or for the creation of Application Profiles with the EAT, the developer can imagine and design the way the service can be used in the future by the end-user. It is important that the design of the service enable the user to easily grasp how to handle the settings of the QoS parameters.

3.2.4 Recommendations for the Feedback on the Status of the Connection of the QoS Categories

Another essential aspect of the availability of a QoS supported Complex Internet Service is the feedback on the status of the connection of the QoS categories chosen by the user. The user will not be willing to bear the additional costs for QoS support, if there is no obvious service in return. It is crucial to visualise the effects of the QoS support, because the difference between an enabled and a disabled QoS service is not always easy to seize for the user.

The format of the feedback should be adapted to the needs of the user, that is to say to his/her profile.

Mediazine expert	Mediazine novice
To verify whether the user settings work properly, the user should be given the possibility to inquire on detailed information on the connection such as e.g. currently transferred amount of data, packet loss, latency etc or to use the feedback of the standard user.	The user should be informed of the correct functioning of the settings by an obvious symbol showing the selected QoS category.

Table 3-3: Recommendations for Mediazine users

The feedback on the status of the connection for the selected QoS category should be delivered by the necessary measurement tools. However, as this part is not a feature of the EAT, the application itself has to care of it.

4 Configuration Guide for EAT

The configuration of the EAT is based on XML files which are stored in a central LDAP database. Due to the fact that the EAT and the application Proxies run in different processes, there are two kinds of settings to be taken.

4.1 EAT Settings

The EAT Settings are mainly for the configuration of the Application Profiles the EAT *refers*. The Application Profiles themselves, however, are *separately* stored within the LDAP database. The EAT configuration file just contains the path to the profiles as well as a list of their unique names (see Appendix, **eat_dresden**.xml as an example).

The following scenario gives a guideline on how to store an Application Profile within the database, and how to configure it for the EAT:

1. Write a new XML file, i.e. the Application Profile for a new application (e.g. **NetMeeting**.xml). Make sure that it is based on the grammar which is specified in ApplicationProfile.dtd (see Appendix).

2. Store this new profile within the LDAP database:

```
java -cp <aquila classes>;<jaxb-rt-1.0-ea.jar> aquila.uil.main.StoreXml
  "cn=NetMeeting,cn=appProfile,cn=eat,cn=rcl" NetMeeting.xml
    aquila.rcl.eat.appProfile.ApplicationProfile
```

3. Add the entry

```
<Profile>NetMeeting.xml</Profile>
```

   to the EAT configuration file of your EAT (within the `ApplicationProfiles` tag).

4. Make sure that the `Path` attribute of `ApplicationProfiles` corresponds to the *basic* path in 2., e.g. `"cn=appProfile,cn=eat,cn=rcl"`.

5. Make sure that the `ACAName` attribute of `EATSettings` contains the correct name of the ACA that belongs to this EAT, e.g. **`"aca_dresden"`**.

6. Store the EAT settings in the LDAP database as well:

```
java -cp <aquila classes>;<jaxb-rt-1.0-ea.jar> aquila.uil.main.StoreXml
  "cn=eat_dresden,cn=eat,cn=rcl" eat_dresden.xml
    aquila.rcl.eat.eatManager.EATSettings
```

7. Start the EAT:

```
java -cp <aquila classes> aquila.rcl.eat.eatManager.EAT eat_dresden
```

4.2 Proxy Settings

Each application Proxy has to have its own configuration file which contains some basic information for the Proxy such as its id, and optionally its full name (or description) and the control port it uses (see Appendix, **sip_dresden**.xml).

Guideline for configuration:

1. Create a new Proxy configuration file, based on ProxySettings.dtd. Make sure that the `EATName` attribute of `ProxySettings` contains the correct name of the EAT that belongs to this Proxy, e.g. **`"eat_dresden"`**.

2. Store the Proxy settings in the LDAP database:

```
java -cp <aquila classes>;<jaxb-rt-1.0-ea.jar> aquila.uil.main.StoreXml
  "cn=sip_dresden,cn=proxy,cn=eat,cn=rcl" sip_dresden.xml
    aquila.rcl.eat.proxy.ProxySettings
```

3. Run the Proxy:

```
java -cp <aquila classes> aquila.rcl.eat.proxy.SIPProxy sip_dresden
```

4. The Proxy registers itself at the already running EAT.

5 Abbreviations

A

ACA	Admission Control Agent
API	Application Programming Interface
AQUILA	Adaptive Resource Control for QoS Using an IP-based Layered Architecture

B

BIA	Basic Internet Application (Legacy Application)

C

CBR	Constant Bit Rate
CIS	Complex Internet Service
Codec	COmpression/DECompression
CORBA	Common Object Request Broker Architecture (see ORB)

D

DSL	Digital Subscriber Line
DTD	Document Type Definition

E

EAT	End-user Application Toolkit

F

FTP	File Transfer Protocol

G

GUI	Graphical User Interface

I

IDL	Interface Definition Language
IP	Internet Protocol
ISDN	Integrated Services Digital Network

J

JDK	Java Development Kit

L

LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol

M

MF	Multi-Field

N

NS	Network Service

O

ORB	Object Request Broker (see CORBA)

P

PR	Peak Rate

Q

QoS	Quality of Service

R

RCL	Resource Control Layer
RSVP	Resource Reservation Protocol

S

SDP	Session Description Protocol
SIP	Session Initiation Protocol
SLA	Service Level Agreement
SLS	Service Level Specification
SR	Sustainable Rate

T

TCP	Transmission Control Protocol

U

UDP	User Datagram Protocol
UML	Unified Modelling Language
URL	Uniform Resource Locator

V

VBR	Variable Bit Rate
VoIP	Voice over IP

X

XML	eXtensible Markup Language

6 References

[D1201] IST-1999-10077-WP1.2-SAG-1201-PU-O/b0, System architecture and specification for the first trial

[D1202] IST-1999-10077-WP1.2-SAG-1202-PU-O/b1, System architecture and specification for the second trial

[D2201] IST-1999-10077-WP2.2-TUD-2201-RE-O/b0, Specification of End-user Application Toolkit

[D2202] IST-1999-10077-WP2.2-TUD-2202-PU-O/b0, Description of user applications for the first trial

[Codecs] PacketCable Audio/Video Codecs Specifications, PKT-SP-CODEC-I02-010620, Cable Television Laboratories, Inc. 2001

[H.323] International Telecommunication Union, “Visual telephone systems and equipment for local area networks which provide a non-guaranteed quality of service”, Recommendation H.323, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, May 1996

[Harold98] E. H. Harold, XML: Extensible Markup Language, IDG Books, 1998

[JAXP] Java API for XML Parsing, http://java.sun.com/xml/

[Jelliffe98] R. Jelliffe, The XML and SGML Cookbook, Prentice Hall, 1998

[SIP] M. Handley, H. Schulzrinne, E. Schooler, J. Rosenberg, “SIP: Session Initiation Protocol”, IETF RFC 2543, March 1999

[XML] Extensible Markup Language (XML), http://www.w3.org/XML/

Appendix

The Application Programming Interface

api.idl

```
/*
    Package aquila.rcl.eat.api
    File    api.idl
    Version 3.7, 12.12.2001

    Dresden University of Technology (TUD),
    National Technical University of Athens (NTUA)

    This file contains the specification of the (internal) api
    of the End-user Application Toolkit (EAT) for the 2nd trial.

    It is intended for the use by the applications as well as the
    Reservation GUI, etc.
*/

#ifndef   api_idl
#define   api_idl

#include "service.idl"
#include "subscriber.idl"
#include "aca.idl"
#include "appProfile.idl"
#include "converter.idl"
#include "eatPersistence.idl"
#include "proxy.idl"
#include "event.idl"

module api {

    /* --- Forward declarations --- */

    interface QoSSessionRequest;
    interface QoSSession;
    interface QoSSessionGroup;
    interface QoSSessionUnit;

    typedef sequence<QoSSession> QoSSessionSeq;
    typedef sequence<QoSSessionUnit> QoSSessionUnitSeq;

    /**
     * The QoSRequester interface should be implemented by the QoS requesting
     * client in or der to have the chance to inform it about some events
     * concerning the requested reservation, e.g. when a provisional reservation
     * has been accepted/rejected by the ACA, or when a reservation brokes, etc.
     */
    typedef event::EventObserver QoSRequester;

    /**
     * APIException containing a message string.
     */
    exception APIException {
        string reason;
    };
```

```
    /* --- Common --- */

    typedef service::ServiceIDSeq ServiceIDSeq;
    typedef sequence<aca::SLS>    slsSeq;
    typedef sequence<aca::Scope>  ScopeSeq;
    typedef sequence<aca::Flow>   FlowSeq;

    // Might become obsolete when new service.idl is available:
    typedef sequence<service::AccountingData> AccountingDataSeq;


    /* --- Login ------------------------------------------------------------ */

    /**
     * The Login interface allows the authentication of an end-user against
     * the EAT (Manager). It is implemented by the EAT.
     * The login information is also forwarded to the ACA.
     */
    interface Login {

        const string SERVER_NAME = "Login";
        const string SERVER_KIND = "EAT login object";

        /**
         * Login at the EAT Manager's API.
         *
         * @param   loginInfo        Account name and password.
         * @param   clientSessionId  Created by a Web server providing the
         *                           Login GUI.
         * @return                   Reference to a new QoSSessionRequest
         *                           object.
         */
        QoSSessionRequest loginClient (
            in subscriber::LoginInfo logInfo,
            in string                clientSessionId)
            raises (APIException);

        /**
         * Gets the reference to the QoSSessionRequest object created at the
         * login.
         * Called by the Reservation GUI, the Regular Reservation GUI, etc.
         *
         * @param   clientSessionId  Web servers' session id, included in the
         *                           URL.
         * @return                   Reference to the existing QoSSessionRequest
         *                           object.
         */
        QoSSessionRequest getQSRequestMeansSessionId (
            in string clientSessionId);
    };


    /* --- QoS Session Request ---------------------------------------------- */

    typedef sequence<string> ClientSessionIdSeq;

    /**
     * The QoSSessionRequest interface is the user agent for QoS session
     * requests and SLA retrieval.
     * Requests can be made on an advanced and on a regular level.
     */
    interface QoSSessionRequest {

        /**
         * The subscriber.
         */
        readonly attribute string accountName;
```

```
        /**
         * All current client session ids.
         */
        readonly attribute ClientSessionIdSeq clientSessionIds;


        /* --- SLAs --- */

        /**
         * The SLAs.
         */
        readonly attribute converter::SLASeq contracts;


        /* --- Advanced Request --- */

        /**
         * For the specification of a (single) advanced request.
         *
         * applicationId     Name that the end-user uses to identify the
         *                   application.
         * serviceComponent  Service component that this reservation corresponds
         *                   to.
         * networkService    Network service id.
         * reqSLS            Requested SLS incl, scope, flow, traffic spec, QoS
         *                   spec, and schedule.
         * proxyId           Id of the proxy to be used (see ApplicationProxy),
         *                   0 for none.
         * enabled           Indicates whether the reservation shall be
         *                   immediately established in the edge router or not.
         */
        struct AdvancedSpec {
            string             applicationId;     // One application
            string             serviceComponent;  // One service component
            service::ServiceID networkService;    // One service
            aca::SLS           reqSLS;            // One SLSpec
            long               proxyId;           // One proxy
            boolean            enabled;
        };

        /**
         * Requests for a (single) QoSSession on advanced level.
         * It is foreseen for reservation requests that are based
         * on the content of the ACA's reservation request interface,
         * the so-called Advanced Reservation Mode.
         *
         * @param  requestSpec  The requested parameters.
         * @param  requester    The requester object that has to be notified
         *                      when something happens with the reservation.
         *                      Can be null.
         * @return              Reference to a new QoSSessionUnit object with
         *                      one single element.
         */
        QoSSessionUnit advancedRequest (
            in AdvancedSpec requestSpec,
            in QoSRequester requester)
            raises (APIException);


        /* --- Advanced Multiple Request --- */

        /**
         * For the specification of a multiple (e.g. bi-directional)
         * advanced request.
         *
         * applicationId     Name that the end-user uses to identify the
```

```
 *                    application.
 * serviceComponent   Service component that this reservation corresponds
 *                    to.
 * networkServices    Network service ids.
 * reqSLSs            Requested SLSs incl. scoped, flows, traffic specs,
 *                    QoS specs, and schedules.
 * proxyId            Id of the proxy to be used (see ApplicationProxy),
 *                    0 for none.
 * enabled            Indicates whether the reservation shall be
 *                    immediately established in the edge router or not.
 */
struct AdvancedMultipleSpec {
    string              applicationId;     // One application
    string              serviceComponent;  // One service component
    ServiceIDSeq        networkServices;   // Several services !
    slsSeq              reqSLSs;           // Several SLSpecs !
    long                proxyId;           // One proxy
    boolean             enabled;
};

/**
 * Advanced request for a multiple (e.g. a bi-directional) reservation,
 * building an inseperable unit of session elements, the session unit.
 *
 * @param  requestSpec  The requested parameters.
 * @param  requester    The requester object that has to be notified
 *                      when something happens with the reservation.
 *                      Can be null.
 * @return              Reference to a new QoSSessionUnit object with
 *                      as many elements as requested. In the case of a
 *                      bi-directional session, there are two request
 *                      elements.
 */
QoSSessionUnit advancedMultipleRequest (
    in AdvancedMultipleSpec requestSpec,
    in QoSRequester         requester)
    raises (APIException);


/* --- Advanced Group Request --- */

/**
 * For the specification of a group request.
 */
typedef sequence<AdvancedMultipleSpec> AdvancedSpecSeq;

/**
 * Advanced request for a group of QoS sessions.
 *
 * @param  groupName     The optional name of the new group (can be "").
 * @param  requestSpecs  An array of (multiple) request specifications.
 * @param  requester     The requester object that has to be notified
 *                       when something happens with the reservation.
 *                       Can be null.
 * @return               An new QoSSessionGroup object, containing
 *                       one QoSSessionUnit object for each (multiple)
 *                       request.
 */
QoSSessionGroup advancedGroupRequest (
    in string          groupName,
    in AdvancedSpecSeq requestSpecs,
    in QoSRequester    requester)
    raises (APIException);


/* --- Regular Request --- */
```

```
        /**
         * Prepares the Regular App GUI with the options from the profile,
         * gets the ids for the session characteristics options to be displayed.
         *
         * @param  applicationProfile  ID of the associated application
         *                             profile.
         * @return                     An array of option ids of the profiles.
         */
        converter::OptionIDSeq prepareSessionCharacteristicsOptions (
            in appProfile::ProfileID applicationProfile)
            raises (APIException);

        /**
         * For the specification of a (single) regular request.
         * Note that for each service component one request is needed.
         *
         * applicationProfile  ID of the associated application profile.
         * applicationId       The application name within the application
         *                     profile.
         * selection           Selected session characteristic option from the
         *                     profile.
         * serviceComponent    The service component that the selection
         *                     corresponds to.
         * reqScope            Scope: reservation style.
         * reqFlow             Source and dest. addresses, ports, protocol,
         *                     DSCP.
         * schedule            Service schedule: reservation time.
         * proxyName           Name of the proxy to be used (see application
         *                     profile), the EAT Manager can look into the *.rc
         *                     file in order to get the id of the proxy;
         *                     "" if no proxy is needed.
         * enabled             Indicates whether the reservation shall be
         *                     immediately established in the edge router or
         *                     not.
         */
        struct RegularSpec {
            appProfile::ProfileID applicationProfile;  // One
            string                applicationId;       //  application
            converter::OptionID   selection;           // One
            string                serviceComponent;    //  service component !!!
            aca::Scope            reqScope;            // One scope
            aca::Flow             reqFlow;             // One flow spec
            aca::ServiceSchedule  schedule;            // One schedule
            string                proxyName;           // One proxy
            boolean               enabled;
        };

        /**
         * Requests for a QoSSession on regular level.
         * It is foreseen for applications which are not QoS-aware
         * but are supported by application profiles for manual,
         * non-professional reservations. The end-user has to ask for the
         * preparation of suitable Session Charactersitics options and requests
         * then for such as QoS session.
         *
         * @param  requestSpec  The requested parameters.
         * @param  requester    The requester object that has to be notified
         *                      when something happens with the reservation.
         *                      Can be null.
         * @return              Reference to a new QoSSessionUnit object
         *                      containing one element for the reservation of
         *                      the specified service component. (For several
         *                      service components, use regularGroupRequest().)
         */
        QoSSessionUnit regularRequest (
            in RegularSpec    requestSpec,
            in QoSRequester requester)
```

```
            raises (APIException);


        /* --- Regular Multiple Request --- */

        /**
         * For the specification of a multiple (e.g. bi-directional) regular
         * request.
         *
         * applicationProfile  ID of the associated application profile.
         * applicationId       The application name within the application
         *                     profile.
         * selection           Selected session characteristic option from the
         *                     profile.
         * serviceComponent    The service component that the selection
         *                     corresponds to.
         * reqScopes           Scopes: reservation styles.
         * reqFlows            Source and dest. addresses, ports, protocol,
         *                     DSCP.
         * schedule            Service schedule: reservation time.
         * proxyName           Name of the proxy to be used (see application
         *                     profile), the EAT Manager can look into the *.rc
         *                     file in order to get the id of the proxy;
         *                     "" if no proxy is needed.
         * enabled             Indicates whether the reservation shall be
         *                     immediately established in the edge router or
         *                     not.
         */
        struct RegularMultipleSpec {
            appProfile::ProfileID applicationProfile;  // One
            string                applicationId;       //  application
            converter::OptionID   selection;           // One
            string                serviceComponent;    //  service component !!!
            ScopeSeq              reqScopes;           // Several scopes !
            FlowSeq               reqFlows;            // Several flow specs !
            aca::ServiceSchedule  schedule;            // One schedule
            string                proxyName;           // One proxy
            boolean               enabled;
        };

        /**
         * Regular request for a multiple (e.g. a bi-directional) reservation,
         * building an inseperable unit of session elements, the session unit.
         *
         * @param  requestSpec  The requested parameters.
         * @param  requester    The requester object that has to be notified
         *                      when something happens with the reservation.
         *                      Can be null.
         * @return              Reference to a new QoSSessionUnit object with
         *                      as many elements as requested. In the case of a
         *                      bi-directional session, there are two request
         *                      elements.
         */
        QoSSessionUnit regularMultipleRequest (
            in RegularMultipleSpec requestSpec,
            in QoSRequester        requester)
            raises (APIException);


        /* --- Regular Group Request --- */

        typedef sequence<RegularMultipleSpec> RegularSpecSeq;

        /**
         * Regular request for a group of QoS sessions,
         * e.g. for different service components per application.
         *
```

```
         * @param  groupName     The optional name of the new group (can be "").
         * @param  requestSpecs  An array of (multiple) request specifications.
         * @param  requester     The requester object that has to be notified
         *                       when something happens with the reservation.
         *                       Can be null.
         * @return               An new QoSSessionGroup object, containing
         *                       one QoSSessionUnit object for each (multiple)
         *                       request.
         */
        QoSSessionGroup regularGroupRequest (
            in string         groupName,
            in RegularSpecSeq requestSpecs,
            in QoSRequester requester)
            raises (APIException);


        /* --- Common --- */

        /**
         * Creates an empty QoSSessionGroup.
         *
         * @param  groupName  The optional name of the new group (can be "").
         * @param  requester  The requester object that has to be notified
         *                    when something happens with the reservation.
         *                    Can be null.
         * @return            An new, empty QoSSessionGroup object.
         */
        QoSSessionGroup createEmptyGroup (
            in string       groupName,
            in QoSRequester requester)
            raises (APIException);

        /**
         * Returns all active reservations.
         *
         * @return  Sequence of QoSSession objects, both groups and units.
         */
        QoSSessionSeq getActiveQoSSessions ();

        /**
         * Returns all active session units, also those which are in goups.
         *
         * @return  Sequence of QoSSessionUnit objects (no groups).
         */
        QoSSessionUnitSeq getQoSSessionUnits ();

        /**
         * Retrives the list of all former and actual reservations of this user.
         *
         * @return  Sequence of ReservationData incl. reservation parameters and
         *          accounting data.
         */
        eatPersistence::ReservationDataSeq retrieveReservationHistory ();

        /**
         * Closes explicitely a client session without logout;
         * removes the client session id from the list.
         *
         * @param   clientSessionId  Web servers' session id, included in the
         *                           URL.
         */
        void close (
            in string clientSessionId)
            raises (APIException);

        /**
         * Logs the end-user out, releases all reservations.
```

```
         */
        void logout ()
            raises (APIException);
    };


    /* --- QoS Session ------------------------------------------------------ */

    /**
     * SessionStatus indicates whether a requested (and by the EAT accepted)
     * reservation is still provisional (waiting for Proxy response) or already
     * accepted by the ACA and therefore active.
     * (Rejected reservations are immediately released, and the client is
     * informed.)
     */
    enum SessionStatus {
        Provisional,      // Waiting for the answer from the Proxy
        Active,           // Requested and admitted by the ACA
        Enabled           // Enabled at the edge router
    };

    /**
     * A QoSSession can be session group or unit.
     */
    interface QoSSession {

        /**
         * The unique session id of this session group or unit.
         */
        readonly attribute long sessionId;

        /**
         * Is this a session group or unit?
         */
        readonly attribute boolean isGroup;

        /**
         * The aggregating group. Can be null.
         */
        readonly attribute QoSSessionGroup group;

        /**
         * The requester of this session, can be null.
         */
        readonly attribute QoSRequester requester;

        /**
         * Releases the associated reservation unit (and all of its elements) or
         * the whole group (all aggregated units and groups are also released).
         * Automatically retrieves and stores the accounting data from the ACA
         * for each unit and all of its elements.
         */
        void release ()
            raises (APIException);

        /**
         * Suspends the associated reservation unit or the whole group,
         * i.e. all reservation units are released but without retrieving
         * any accounting information.
         */
        void suspend ()
            raises (APIException);
    };

    /**
     * The QoSSessionGroup interface belongs to a reservation group,
     * e.g. reservations (units) for several service components per application.
```

```
 */
interface QoSSessionGroup : QoSSession {

    /**
     * The optional name of this group.
     */
    attribute string groupName;

    /**
     * The QoS sessions. Can be either other essions groups or units or
     * even both.
     */
    readonly attribute QoSSessionSeq sessions;

    /**
     * Returns one specific QoSSession object of this group.
     *
     * @param  sessionId  The session id of the reservation.
     * @return            The QoSSession object with the specified id.
     *                    Null if none.
     */
    QoSSession getSession (in long sessionId);

    /**
     * Adds/joins an already established, QoSSession to this group.
     *
     * @param  sessionId  The session to be added to this group.
     */
    void join (in QoSSession session)
        raises (APIException);

    /**
     * Removes a QoSSession object from this group without releasing it.
     *
     * @param  sessionId  The session to be removed from this group.
     */
    void leave (in QoSSession session)
        raises (APIException);

    /**
     * Reamoves all QoSSession objects from this group without releasing
     * them.
     */
    void leaveAll ()
        raises (APIException);
};

/**
 * The QoSSessionUnit interface belongs to an actual reservation unit
 * (in terms of the ACA: a reservation bundle). While such a unit may
 * consist of more than one reservation element, it must be handled as only
 * one (multiple) reservation, for example a bi-directional one.
 */
interface QoSSessionUnit : QoSSession {

    readonly attribute short numberOfElements; // e.g. 1 for an uni-
                                               // directional reservation,
                                               // 2 for a bidirectional one
    readonly attribute SessionStatus status;

    // Content of the advanced/regular request:
    readonly attribute appProfile::ProfileID applicationProfile;//"" if none
    readonly attribute string                applicationId;
    readonly attribute converter::OptionID   selection;         //"" if none
    readonly attribute string                serviceComponent;
    readonly attribute ServiceIDSeq          networkServices;
    readonly attribute slsSeq                reqSLSs;
```

```
        readonly attribute long                 proxyId;

        /**
         * Gets the accounting data of this session.
         *
         * @return  Accounting information array of this session unit,
         *          containing one entry per requested reservation element.
         */
        AccountingDataSeq getAccountingInformation ();

        /**
         * Enables an already active but not in the edge router installed
         * reservation.
         */
        void enable ()
            raises (APIException);
    };


    /* --- RequestEvent (for QoSRequester) ---------------------------------- */

    /**
     * RequestKind describes what with a reservation (request) ca be happen:
     * A provisional reservation can be accepted or rejected.
     * An already established reservation can be released, e.g. when it brokes
     * or when the schedule finishes.
     */
    enum RequestKind {
        Accepted,
        Rejected,
        Released
    };

    /**
     * RequestEvent contains the QoSSession id, the kind of the occuring event,
     * and a optional reason.
     */
    valuetype RequestEvent : event::GeneralEvent {
        public long        sessionId;
        public RequestKind kind;
        public string      reason;
    };


    /* --- Application Manager ---------------------------------------------- */

    /**
     * ApplicationInformation contains the ID of the associated app profile,
     * the app's name, the version, and the build no.
     */
    struct ApplicationInformation {
        appProfile::ProfileID applicationProfile;
        string                applicationName;
        string                versionNo;
        string                buildNo;
    };

    typedef sequence<ApplicationInformation> ApplicationInformationSeq;

    /**
     * The ApplicationManager interface provides information about installed
     * available application (profiles), application proxies, etc.
     */
    interface ApplicationManager {

        const string SERVER_NAME = "ApplicationManager";
        const string SERVER_KIND = "EAT application manager object";
```

```
        /**
         * Gets all available apps for the Legacy App GUI.
         *
         * @return  Sequence of ApplicationInformation objects.
         */
        ApplicationInformationSeq getAvailableApps ();

        /**
         * Gets all installed proxies for the Reservation GUI.
         *
         * @return  Sequence of ApplicationProxy objects.
         */
        proxy::ProxyDescriptionSeq getApplicationProxies ();
    };


    /* --- Network Service Distributor ------------------------------------ */

    /**
     * NetworkService is an "mirror" of the Network Service Profile, accessible
     * via CORBA. (An Application does not have access to the LDAP DB.)
     */
    struct NetworkService {
        service::ServiceID          servId;
        string                      fullName;
        service::QoSSpec            defaultQoS;
        service::ReservationStyleSeq styles;
    };
    typedef sequence<NetworkService> NetworkServiceSeq;

    /**
     * The ServiceDistributor interface provides information about available
     * network services, and which ones are included in the current SLA.
     */
    interface ServiceDistributor {

        const string SERVER_NAME = "ServiceDistributor";
        const string SERVER_KIND = "EAT service distributor object";

        /**
         * Gets all available network services.
         *
         * @return  Sequence of NetworkService objects.
         */
        NetworkServiceSeq getNetworkServices ();
    };
};

#endif // api_idl
```

SampleClient.java

```
/**
 * Title:        AQUILA, RCL, EAT<p>
 * Description:  Sample API Client<p>
 * Copyright:    Copyright (c) Falk Fünfstück<p>
 * Company:      TUD<p>
 * @author       Falk Fünfstück
 * @version      1.0
 */

package aquila.rcl.eat.gui;

import aquila.rcl.aca.*;
```

```
import aquila.rcl.eat.api.*;
import aquila.rcl.eat.api.QoSSessionRequestPackage.*;
import aquila.rcl.service.*;
import aquila.rcl.subscriber.LoginInfo;
import aquila.rcl.tc.TrafficSpec;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import org.omg.CORBA.ORB;
import org.omg.CORBA.SystemException;
import org.omg.CORBA.UserException;
import org.omg.CosNaming.NameComponent;
import org.omg.CosNaming.NamingContext;
import org.omg.CosNaming.NamingContextHelper;
import org.omg.CosNaming.NamingContextPackage.AlreadyBound;
import java.util.Properties;
import javax.naming.Context;
import javax.naming.directory.DirContext;

/**
 * This class provides a very simple sample scenario on how to use the EAT API.
 */
public class SampleClient {

    private static SampleClient sc;
    private ORB                 orb  = null;
    private NamingContext       nctx = null;

    public SampleClient(String[] argv) {

        Properties props = new Properties(System.getProperties());
        try  {
            String filename;
            filename = props.getProperty(
                "aquila.rcl.propertyFile", "aquila.rc");
            filename = new File(filename).getAbsolutePath();
            props.load(new FileInputStream(filename));
        }
        catch (java.io.IOException ioex) {
            // just ignore
        };

        orb = ORB.init(argv, props);

        if (orb == null) {
            System.err.println("Could not get the ORB");
            System.exit(1);
        };

        try {
            NamingContext nameService = NamingContextHelper.narrow(
                orb.resolve_initial_references("NameService"));

            NameComponent ncAQUILA = new NameComponent("aquila", "directory");
            NameComponent ncRCL    = new NameComponent("rcl",    "directory");

            NameComponent[] nameAQUILA = {ncAQUILA};
            try {
                nameService.bind_new_context(nameAQUILA);
            }
            catch (AlreadyBound ab) {};

            NameComponent[] name = {ncAQUILA, ncRCL};
            try {
                nameService.bind_new_context(name);
            }
```

```
            catch (AlreadyBound ab) {};

            nctx = NamingContextHelper.narrow(nameService.resolve(name));
            }
        catch (UserException exc) {
            System.err.println(
                "UserException: could not resolve aquila.rcl");
            System.exit(1);
        }
            catch (SystemException exc) {
            System.err.println(
                "SystemException: could not resolve aquila.rcl: " + exc);
            System.exit(1);
        };
    };

    protected void start(String eatName) {

        try {

            // --- Local EAT ------------------------------------------------

            Login theLogin = null;
            ApplicationManager theAppManager = null;
            ServiceDistributor theServiceDistributor = null;

            NameComponent[] nameLogin = {
                new NameComponent(eatName, "directory"),
                new NameComponent(theLogin.SERVER_NAME, theLogin.SERVER_KIND)
            };
            theLogin = LoginHelper.narrow(nctx.resolve(nameLogin));

            System.out.println("Got the login object");

            NameComponent[] nameAppManager = {
                new NameComponent(eatName, "directory"),
                new NameComponent(
                    theAppManager.SERVER_NAME, theAppManager.SERVER_KIND)
            };
            theAppManager = ApplicationManagerHelper.narrow(
                nctx.resolve(nameAppManager));

            System.out.println("Got the app manager");

            NameComponent[] nameServiceDistributor = {
                new NameComponent(eatName, "directory"),
                new NameComponent(
                    theServiceDistributor.SERVER_NAME,
                    theServiceDistributor.SERVER_KIND)
            };
            theServiceDistributor = ServiceDistributorHelper.narrow(
                    nctx.resolve(nameServiceDistributor));

            System.out.println("Got the service distributor");


            // --- Tests ----------------------------------------------------

            // Network services:
            NetworkService[] services = new NetworkService[0];

            System.out.println("Available network services:");
            services = theServiceDistributor.getNetworkServices();
            for (int i = 0; i < services.length; i++) {
                System.out.println(services[i].servId);
                System.out.println(services[i].fullName);
            };
```

```
        // Available Apps:
        ApplicationInformation[] apps = new ApplicationInformation[0];
        System.out.println("Available Applications:");
        apps = theAppManager.getAvailableApps();
        for (int i = 0; i < apps.length; i++) {
            System.out.println(apps[i].applicationProfile);
            System.out.println(apps[i].applicationName);
        };

        // Login:
        System.out.println("Try to login 'wi' ...");
        QoSSessionRequest request = theLogin.loginClient(
            new LoginInfo("wi", "geheim"), "0815");

        // Request:
        if (request != null) {
            System.out.println("Got the request object");

            // 1. try: PMC

            // Structs which contains object references have to be
            // initialized, because their default constructor is empty!
            String app = "app1";
            String com = "component1";
            String ns  = "PMC";

            Scope s = new Scope("p2a");
            Flow f = new Flow(
                new Address("10.0.5.2", "255.255.255.0", 0, 0),
                new Address("-1", "-1", 0, 0), (short) 0, (short) 0);
            TrafficSpec ts = new TrafficSpec(
            //  PR     BSP   SR     BSS   m   M    EAR PR1 PR2
                10000, 2048, 8500, 2048, 40, 512, -1, -1, -1 );
            QoSSpec qos = new QoSSpec(100, 10, 0, 50, false);
            ServiceSchedule ss = new ServiceSchedule("", 0, 0, 0);

            SLS slspec = new SLS(s, f, ts, qos, ss);

            AdvancedSpec as = new AdvancedSpec(
                app, com, ns, slspec, 0, true);

            System.out.println("Try to request ...");

            QoSSession session = null;
            session = request.advancedRequest(as, null);

            if (session != null) {
                System.out.println("Got session: " +
                    session.sessionId() + " for NS: " + ns);
                session.release();
                System.out.println("Session released");
            }
            else {
                System.out.println("Couldn't get session");
            };

            request.logout();
        }
        else {
            System.out.println("Couldn't get request");
        };

    }
    catch (APIException apiExc) {
        System.out.println(apiExc.reason);
    }
```

```
        catch (Exception exc) {
            System.out.println("Catched exception: " + exc);
        };

        System.exit(0);
    };

    public static void main (String[] argv) {

        if (argv.length < 1) {
            System.out.println();
            System.out.println("Usage: SampleClient <eat_name>");
            System.exit(1);
        };

        sc = new SampleClient(argv);
        sc.start(argv[argv.length - 1]);
    };
};
```

Application Profile

ApplicationProfile.dtd

```
<!--
        The ApplicationProfile gives the possibility to describe an
        application in detail in the scope of QoS mechanisms (QoS offer
        and request) towards a QoS enabled network. An ApplicationProfile
        collects protocol information (to know which ports are used, etc.)
        and implementation information especially codec information.
        The implementation information describes the QoSRequirements of
        the codec/service components, the traffic produced by the codec,
        and gives user-friendly descriptions of the different possible
        quality levels.
        This information enables:
        1. the presentation towards the end-user of the application
        quality levels
        2. the request for QoS (QoS requirements and produced traffic
        behaviour)
        3. the connection to the network layer of the QoS enables network
 -->
<!ELEMENT ApplicationProfile (CodecImplementation+, protocol*)>
<!ATTLIST ApplicationProfile
 name CDATA #REQUIRED
 version CDATA #REQUIRED
 build CDATA #IMPLIED
 type (VoIP | MULTIMEDIA | STREAMINGVIDEO | STREAMINGAUDIO | GAME |
 OTHER) #REQUIRED
 scope (unspecified | unidirectional | bidirectional | p2p |
 xdirectional) #REQUIRED
>

<!ELEMENT CodecImplementation (Codec,TransportProtocol)>
<!--
        As many applications implement the same codecs, the application
        profile refer to them.
        A transport protocol is associated to each implemented codec.
 -->
        <!ELEMENT Codec (#PCDATA)>
        <!ATTLIST Codec
         xlink:form CDATA #FIXED "simple"
         href CDATA #REQUIRED
        >
        <!ELEMENT TransportProtocol EMPTY>
```

```
        <!ATTLIST TransportProtocol
                name (TCP | UDP) "TCP"
        >
<!ELEMENT protocol (lowerPortNo?,upperPortNo?,isControlPort?)>
<!ATTLIST protocol
 name (unspecified | RTP | RTSP | RSVP | SIP | SDP | H320 | H321 |
 H322 | H323 | H324) "unspecified"
>
        <!ELEMENT lowerPortNo (#PCDATA)>
        <!ATTLIST lowerPortNo
         value (unspecified | fixed | configurable) "fixed"
        >
        <!ELEMENT upperPortNo (#PCDATA)>
        <!ATTLIST upperPortNo
         value (unspecified | fixed | configurable) "fixed"
        >
        <!ELEMENT isControlPort (#PCDATA)>
        <!ATTLIST isControlPort
         value (unspecified | yes | no) "no"
        >
<!ELEMENT description (#PCDATA)>
```

CodecSpecification.dtd (Codec Profile)

```
<!ELEMENT CodecSpecification  (QoSRequirement, Option+)>
<!ATTLIST CodecSpecification
        name CDATA #REQUIRED
>
<!--
    The QoSRequirement part of the CodecSpecification corresponds
    to the general QoS requirements of the implemented codec under
    whose the codec/service component can work properly.
 -->
<!ELEMENT QoSRequirement (maxDelay, maxJitter, maxLoss, bwGuarantee,
ordering)>
<!--
    maxDelay    : "one way latency as unit milliseconds"
    maxJitter   : "delay variation as unit milliseconds"
    maxLoss     : "packet loss probability as unit percent"
    bwGuarantee : "percentage of bandwidth that should be guaranteed"
    ordering    : "Must the packets be ordered?"
-->
    <!ELEMENT maxDelay (#PCDATA)>
    <!ATTLIST maxDelay
        unit CDATA #FIXED "ms"
        value (unspecified | veryLow | low | medium | high | veryHigh
        | notRelevant) "unspecified"
    >
    <!ELEMENT maxJitter (#PCDATA)>
    <!ATTLIST maxJitter
        unit CDATA #FIXED "ms"
        value (unspecified | veryLow | low | medium | high | veryHigh
        | notRelevant) "unspecified"
    >
    <!ELEMENT maxLoss (#PCDATA)>
    <!ATTLIST maxLoss
        unit CDATA #FIXED "%"
        value (unspecified | veryLow | low | medium | high | veryHigh
        | notRelevant) "unspecified"
    >
    <!ELEMENT bwGuarantee (#PCDATA)>
    <!ATTLIST bwGuarantee
        unit CDATA #FIXED "%"
        requirement (unspecified | veryLow | low | medium | high |
        veryHigh | notRelevant) "unspecified"
```

```
    >
    <!ELEMENT ordering EMPTY>
    <!ATTLIST ordering
        value (unspecified | true | false) "true"
    >

<!--
    An Option corresponds to a "quality" level that is proposed by
    the implementation of the codec, e.g. picture       size, sound
    quality. An Option consists of the generic so called
    SessionCharacteristics (the characteristics in the end-user
    language) and TrafficSpecification (describing the traffic
    produced by the application in qualitative and      quantitative
    terms, and the AQUILA dependent AQUILASpecification (technical
    information related to the NSs)
 -->
<!ELEMENT Option (SessionCharacteristic+, TrafficSpecification?,
AQUILASpecification?)>
<!ATTLIST Option
    optionID CDATA #REQUIRED
    serviceComponent (AUDIO | SPEECH | VIDEO | DATA) #REQUIRED
    description CDATA #IMPLIED
>
    <!ELEMENT SessionCharacteristic (name, semanticalGroup*)>
    <!--
        name                    : of the service component (e.g. picture size)
        qualifier       : adjective, terms describing the service
        component (e.g. medium, 4CIF ...)
     -->
        <!ELEMENT name (#PCDATA)>
        <!ELEMENT semanticalGroup (description?, qualifier*)>
        <!ATTLIST semanticalGroup
                        type (Technical | UserFriendly) "UserFriendly"
        >
            <!ELEMENT description (#PCDATA)>

    <!ELEMENT TrafficSpecification  (type+, duration, adaptivity,
    burstiness, packetSize, bitRate, flow)>
    <!--
        type       : "type of the traffic"
        duration   : "living time of the traffic"
        adaptivity : "adaptivity of the traffic to the capacity of the
        connection (application level QoS adaptation)"
        burstiness : "burstiness of the traffic"
        packetSize : "size ofthe packet"
        bitRate    : "bit rate"
        flow       : "greediness of the flow"
    -->
        <!ELEMENT type EMPTY>
        <!ATTLIST type
            type (unspecified | realTime | nonRealTime | stream | elastic)
            "unspecified"
        >
        <!ELEMENT duration EMPTY>
        <!ATTLIST duration
            value (unspecified | shortLiving | longLiving )"unspecified"
        >
        <!ELEMENT adaptivity EMPTY>
        <!ATTLIST adaptivity
            value (unspecified | yes | no) "unspecified"
        >
        <!ELEMENT burstiness EMPTY>
        <!ATTLIST burstiness
            value (unspecified | yes | no) "unspecified"
        >
        <!ELEMENT packetSize (averagePacketSize?, maximumPacketSize?,
        minimumPolicedUnit?)>
```

```
        <!ATTLIST packetSize
            variability (unspecified | constant | variable) "constant"
            qualifier (unspecified | verySmall | small | medium | big
            | veryBig) "medium"
                >
        <!ELEMENT averagePacketSize (#PCDATA)>
        <!ATTLIST averagePacketSize
            unit CDATA #FIXED "bytes"
            qualifier (unspecified | verySmall | small | medium | big
            | veryBig) "medium"
        >
        <!ELEMENT maximumPacketSize (#PCDATA)>
        <!ATTLIST maximumPacketSize
            unit CDATA #FIXED "bytes"
            qualifier (unspecified | verySmall | small | medium | big
            | veryBig) "medium"
        >
        <!ELEMENT minimumPolicedUnit (#PCDATA)>
        <!ATTLIST minimumPolicedUnit
            unit CDATA #FIXED "bytes"
            qualifier (unspecified | verySmall | small | medium | big
            | veryBig) "medium"
        >
        <!ELEMENT bitRate (peakRate?, averageRate)>
        <!ATTLIST bitRate
            variability (constant | variable) "constant"
            qualifier (unspecified | verySmall | small | medium | big
            | veryBig) "medium"
        >
        <!ELEMENT peakRate (#PCDATA)>
        <!ATTLIST peakRate
            unit CDATA #FIXED "bit/s"
            qualifier (unspecified | veryLow | low | medium | high |
            veryHigh) "medium"
        >
        <!ELEMENT averageRate (#PCDATA)>
        <!ATTLIST averageRate
            unit CDATA #FIXED "bit/s"
            qualifier (unspecified | veryLow | low | medium | high |
            veryHigh) "medium"
        >
        <!ELEMENT flow EMPTY>
        <!ATTLIST flow
            value (unspecified | greedy | non-greedy) "unspecified"
        >

    <!ELEMENT AQUILASpecification (serviceID?, BSP, BSS, minPU, maxPS,
    PR, SR, EAR, PR1, PR2)>
    <!--
        serviceID : Name of the AQUILA NS
        BSP       : bucket size for PR (bytes)
        BSS       : bucket size for SR (bytes)
        minPU     : minimum policed unit (bytes)
        maxPS     : maximum (allowed) packet size (bytes)
        PR        : peak rate (bit/s)
        SR        : sustainable rate (bit/s)
        EAR       : Expected Average Rate (bit/s) - Not used
        PR1       : first threshold for bilevel (bit/s) - Not used
        PR2       : second threshold for bilevel (bit/s) - Not used
    -->
        <!ELEMENT serviceID EMPTY>
        <!ATTLIST serviceID
            value (PCBR | PVBR | PMM | PMC | STD | CUSTOM) "STD"
        >
        <!ELEMENT BSP (#PCDATA)>
        <!ATTLIST BSP
            unit CDATA #FIXED "bytes"
```

```
        >
        <!ELEMENT BSS (#PCDATA)>
        <!ATTLIST BSS
            unit CDATA #FIXED "bytes"
        >
        <!ELEMENT minPU (#PCDATA)>
        <!ATTLIST minPU
            unit CDATA #FIXED "bytes"
        >
        <!ELEMENT maxPS (#PCDATA)>
        <!ATTLIST maxPS
            unit CDATA #FIXED "bytes"
        >
        <!ELEMENT PR (#PCDATA)>
        <!ATTLIST PR
            unit CDATA #FIXED "bit/s"
        >
        <!ELEMENT SR (#PCDATA)>
        <!ATTLIST SR
            unit CDATA #FIXED "bit/s"
        >
        <!ELEMENT EAR (#PCDATA)>
        <!ATTLIST EAR
            unit CDATA #FIXED "bit/s"
        >
        <!ELEMENT PR1 (#PCDATA)>
        <!ATTLIST PR1
            unit CDATA #FIXED "bit/s"
        >
        <!ELEMENT PR2 (#PCDATA)>
        <!ATTLIST PR2
            unit CDATA #FIXED "bit/s"
        >

<!ELEMENT value (#PCDATA)>
<!ELEMENT unit (#PCDATA)>
<!ELEMENT requirement (#PCDATA)>
<!ELEMENT qualifier (#PCDATA)>
```

Example: Codec G_723_1.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE CodecSpecification PUBLIC "G.723.1"
"file:D:\ProfileAQUILAv8\CodecSpecificationV1.dtd">
<CodecSpecification name="G.723.1">
     <QoSRequirement>
        <maxDelay value="low" unit="ms">250</maxDelay>
        <maxJitter value="veryLow" unit="ms">30</maxJitter>
        <maxLoss value="unspecified" unit="%">10</maxLoss>
        <bwGuarantee unit="%" requirement="low"/>
        <ordering value="true"/>
     </QoSRequirement>
     <Option serviceComponent="SPEECH" optionID="1" description="Speech
     component">
        <SessionCharacteristic>
                <name>Speaking mode</name>
                <semanticalGroup type="UserFriendly">
                        <qualifier>One speaker speaks at a time. Simultaneous
                        speaking of more than one spaeker is not possible.
                        Speakers have to speak one after the other.
                        </qualifier>
                </semanticalGroup>
                <semanticalGroup type="Technical">
                        <qualifier>Half-duplex</qualifier>
                </semanticalGroup>
```

```
        </SessionCharacteristic>
        <TrafficSpecification>
            <type type="realTime">
            </type>
            <duration value="longLiving">
            </duration>
            <adaptivity value="unspecified">
            </adaptivity>
            <burstiness value="no">
            </burstiness>
            <packetSize variability="constant" qualifier="medium">
                <averagePacketSize unit="bytes" qualifier="medium">
                </averagePacketSize>
                <maximumPacketSize unit="bytes" qualifier="medium">
                </maximumPacketSize>
                <minimumPolicedUnit unit="bytes" qualifier="medium">
                </minimumPolicedUnit>
            </packetSize>
            <bitRate variability="constant" qualifier="medium">
                <peakRate unit="bit/s" qualifier="medium">
                </peakRate>
                <averageRate unit="bit/s" qualifier="medium">
                </averageRate>
            </bitRate>
            <flow value="unspecified">
            </flow>
        </TrafficSpecification>
        <AQUILASpecification>
                <serviceID value="PCBR"/>
                <BSP unit="bytes">2000</BSP>
                <BSS unit="bytes">-</BSS>
                <minPU unit="bytes">78</minPU>
                <maxPS unit="bytes">-</maxPS>
                <PR unit="bit/s">60000</PR>
                <SR unit="bit/s">-</SR>
                <EAR unit="bit/s">20000</EAR>
                <PR1 unit="bit/s">-</PR1>
                <PR2 unit="bit/s">-</PR2>
        </AQUILASpecification>
    </Option>
    <Option serviceComponent="SPEECH" optionID="2">
        <SessionCharacteristic>
                <name>Speaking mode</name>
                <semanticalGroup type="UserFriendly">
                        <qualifier>The speakers can speak simutaneously during
                        the communication.</qualifier>
                </semanticalGroup>
                <semanticalGroup type="Technical">
                        <qualifier>full-duplex</qualifier>
                </semanticalGroup>
        </SessionCharacteristic>
        <TrafficSpecification>
            <type type="realTime">
            </type>
            <duration value="longLiving">
            </duration>
            <adaptivity value="unspecified">
            </adaptivity>
            <burstiness value="no">
            </burstiness>
            <packetSize variability="constant" qualifier="medium">
                <averagePacketSize unit="bytes" qualifier="medium">
                </averagePacketSize>
                <maximumPacketSize unit="bytes" qualifier="small">
                </maximumPacketSize>
                <minimumPolicedUnit unit="bytes" qualifier="medium">
                </minimumPolicedUnit>
```

```
            </packetSize>
            <bitRate variability="constant" qualifier="medium">
                <peakRate unit="bit/s" qualifier="medium">
                </peakRate>
                <averageRate unit="bit/s" qualifier="medium">
                </averageRate>
            </bitRate>
            <flow value="unspecified">
            </flow>
        </TrafficSpecification>
<AQUILASpecification>
<BSP unit="bytes">2000</BSP>
<BSS unit="bytes">-</BSS>
<minPU unit="bytes">78</minPU>
<maxPS unit="bytes">-</maxPS>
<PR unit="bit/s">90000</PR>
<SR unit="bit/s">-</SR>
<EAR unit="bit/s">400000</EAR>
<PR1 unit="bit/s">-</PR1>
<PR2 unit="bit/s">-</PR2>
</AQUILASpecification>
</Option>
</CodecSpecification>
```

EAT Script

EATScript.dtd

```
<!ELEMENT EATScript (Import?,(Login|Request|Release|Logout)*)>
<!ELEMENT Import (ImportLogin,ImportRequest*)*>
<!ELEMENT ImportLogin EMPTY>
<!ATTLIST ImportLogin
 LoginName ID #REQUIRED
 Password CDATA #REQUIRED
>
<!ELEMENT ImportRequest EMPTY>
<!ATTLIST ImportRequest
 LoginName IDREF #REQUIRED
 RequestId ID #REQUIRED
>
<!ELEMENT Login EMPTY>
<!ATTLIST Login
 LoginName ID #REQUIRED
 Password CDATA #REQUIRED
>
<!ELEMENT Request (RequestSpec+)>
<!ATTLIST Request
 LoginName IDREF #REQUIRED
 RequestId ID #REQUIRED
 ProxyId CDATA #IMPLIED
>
<!ELEMENT RequestSpec (NetworkService,SLS)>
<!ELEMENT NetworkService EMPTY>
<!ATTLIST NetworkService
 ServiceId (PCBR | PVBR | PMM | PMC | STD | CUSTOM) "STD"
>
<!ELEMENT SLS (Scope,Flow,TrafficSpec,Schedule)>
<!ELEMENT Scope EMPTY>
<!ATTLIST Scope
 ReservationStyle (p2p | p2a | p2m | a2p) "p2p"
>
<!ELEMENT Flow (Source,Destination,ProtocolId,DSCP)>
<!ELEMENT TrafficSpec (PR,BSP,SR,BSS,m,M,EAR,PR1,PR2)>
<!ELEMENT Schedule (DateTime,Duration,Idle,Cycles)>
```

```
<!ELEMENT Source (Address)>
<!ELEMENT Destination (Address)>
<!ELEMENT ProtocolId (#PCDATA)>
<!ELEMENT DSCP (#PCDATA)>
<!ELEMENT Address (IPAddress,NetMask,LowerPortNumber,UpperPortNumber)>
<!ELEMENT IPAddress (#PCDATA)>
<!ELEMENT NetMask (#PCDATA)>
<!ELEMENT LowerPortNumber (#PCDATA)>
<!ELEMENT UpperPortNumber (#PCDATA)>
<!ELEMENT PR (#PCDATA)>
<!ATTLIST PR
 Unit CDATA #FIXED "bit/s"
>
<!ELEMENT BSP (#PCDATA)>
<!ATTLIST BSP
 Unit CDATA #FIXED "bytes"
>
<!ELEMENT SR (#PCDATA)>
<!ATTLIST SR
 Unit CDATA #FIXED "bit/s"
>
<!ELEMENT BSS (#PCDATA)>
<!ATTLIST BSS
 Unit CDATA #FIXED "bytes"
>
<!ELEMENT m (#PCDATA)>
<!ATTLIST m
 Unit CDATA #FIXED "bytes"
>
<!ELEMENT M (#PCDATA)>
<!ATTLIST M
 Unit CDATA #FIXED "bytes"
>
<!ELEMENT EAR (#PCDATA)>
<!ATTLIST EAR
 Unit CDATA #FIXED "bit/s"
>
<!ELEMENT PR1 (#PCDATA)>
<!ATTLIST PR1
 Unit CDATA #FIXED "bit/s"
>
<!ELEMENT PR2 (#PCDATA)>
<!ATTLIST PR2
 Unit CDATA #FIXED "bit/s"
>
<!ELEMENT DateTime (#PCDATA)>
<!ELEMENT Duration (#PCDATA)>
<!ELEMENT Idle (#PCDATA)>
<!ELEMENT Cycles (#PCDATA)>
<!ELEMENT Release EMPTY>
<!ATTLIST Release
 RequestId IDREF #REQUIRED
>
<!ELEMENT Logout EMPTY>
<!ATTLIST Logout
 LoginName IDREF #REQUIRED
>
```

Example.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE EATScript SYSTEM "EATScript.dtd">
<EATScript>
    <!-- Login user wi: -->
    <Login LoginName="wi" Password="geheim"/>
```

```
<!-- First request is for PMC: -->
<Request LoginName="wi" RequestId="wi01">
    <RequestSpec>
        <NetworkService ServiceId="PMC"/>
        <SLS>
            <Scope ReservationStyle="p2a"/>
            <Flow>
                <Source>
                    <Address>
                        <IPAddress>192.168.94.0</IPAddress>
                        <NetMask>255.255.255.0</NetMask>
                        <LowerPortNumber>0</LowerPortNumber>
                        <UpperPortNumber>0</UpperPortNumber>
                    </Address>
                </Source>
                <Destination>
                    <Address>
                        <IPAddress>-1</IPAddress>
                        <NetMask>-1</NetMask>
                        <LowerPortNumber>0</LowerPortNumber>
                        <UpperPortNumber>0</UpperPortNumber>
                    </Address>
                </Destination>
                <ProtocolId>0</ProtocolId>
                <DSCP>0</DSCP>
            </Flow>
            <TrafficSpec>
                <PR Unit="bit/s">100</PR>
                <BSP Unit="bytes">1024</BSP>
                <SR Unit="bit/s">50</SR>
                <BSS Unit="bytes">1000</BSS>
                <m Unit="bytes">40</m>
                <M Unit="bytes">512</M>
                <EAR Unit="bit/s">-1</EAR>
                <PR1 Unit="bit/s">-1</PR1>
                <PR2 Unit="bit/s">-1</PR2>
            </TrafficSpec>
            <Schedule>
                <DateTime>0</DateTime>
                <Duration>0</Duration>
                <Idle>0</Idle>
                <Cycles>0</Cycles>
            </Schedule>
        </SLS>
    </RequestSpec>
</Request>
<!-- Second request is for PMM (bidirectional): -->
<Request LoginName="wi" RequestId="wi02" ProxyId="1">
    <RequestSpec>
        <NetworkService ServiceId="PMM"/>
        <SLS>
            <Scope ReservationStyle="p2p"/>
            <Flow>
                <Source>
                    <Address>
                        <IPAddress>192.168.94.0</IPAddress>
                        <NetMask>255.255.255.0</NetMask>
                        <LowerPortNumber>0</LowerPortNumber>
                        <UpperPortNumber>0</UpperPortNumber>
                    </Address>
                </Source>
                <Destination>
                    <Address>
                        <IPAddress>192.168.95.0</IPAddress>
                        <NetMask>255.255.255.0</NetMask>
                        <LowerPortNumber>0</LowerPortNumber>
                        <UpperPortNumber>0</UpperPortNumber>
```

```
                    </Address>
                </Destination>
                <ProtocolId>0</ProtocolId>
                <DSCP>0</DSCP>
            </Flow>
            <TrafficSpec>
                <PR Unit="bit/s">-1</PR>
                <BSP Unit="bytes">-1</BSP>
                <SR Unit="bit/s">1000</SR>
                <BSS Unit="bytes">-1</BSS>
                <m Unit="bytes">40</m>
                <M Unit="bytes">512</M>
                <EAR Unit="bit/s">-1</EAR>
                <PR1 Unit="bit/s">-1</PR1>
                <PR2 Unit="bit/s">-1</PR2>
            </TrafficSpec>
            <Schedule>
                <DateTime>0</DateTime>
                <Duration>0</Duration>
                <Idle>0</Idle>
                <Cycles>0</Cycles>
            </Schedule>
        </SLS>
    </RequestSpec>
    <RequestSpec>
        <NetworkService ServiceId="PMM"/>
        <SLS>
            <Scope ReservationStyle="p2p"/>
            <Flow>
                <Source>
                    <Address>
                        <IPAddress>192.168.95.0</IPAddress>
                        <NetMask>255.255.255.0</NetMask>
                        <LowerPortNumber>0</LowerPortNumber>
                        <UpperPortNumber>0</UpperPortNumber>
                    </Address>
                </Source>
                <Destination>
                    <Address>
                        <IPAddress>192.168.94.0</IPAddress>
                        <NetMask>255.255.255.0</NetMask>
                        <LowerPortNumber>0</LowerPortNumber>
                        <UpperPortNumber>0</UpperPortNumber>
                    </Address>
                </Destination>
                <ProtocolId>0</ProtocolId>
                <DSCP>0</DSCP>
            </Flow>
            <TrafficSpec>
                <PR Unit="bit/s">-1</PR>
                <BSP Unit="bytes">-1</BSP>
                <SR Unit="bit/s">1000</SR>
                <BSS Unit="bytes">-1</BSS>
                <m Unit="bytes">40</m>
                <M Unit="bytes">512</M>
                <EAR Unit="bit/s">-1</EAR>
                <PR1 Unit="bit/s">-1</PR1>
                <PR2 Unit="bit/s">-1</PR2>
            </TrafficSpec>
            <Schedule>
                <DateTime>0</DateTime>
                <Duration>0</Duration>
                <Idle>0</Idle>
                <Cycles>0</Cycles>
            </Schedule>
        </SLS>
    </RequestSpec>
```

```
    </Request>
    <!-- Release the first reservation: -->
    <Release RequestId="wi01"/>
</EATScript>
```

Example1.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE EATScript SYSTEM "EATScript.dtd">
<EATScript>
    <!-- Re-login and re-identify the second request: -->
    <Import>
        <ImportLogin LoginName="wi" Password="geheim"/>
        <ImportRequest LoginName="wi" RequestId="wi02"/>
    </Import>
    <!-- Release the second reservation: -->
    <Release RequestId="wi02"/>
    <!-- May be logout the user -->
    <Logout LoginName="wi"/>
</EATScript>
```

EAT & Proxy configuration

EATSettings.dtd

```
<!ELEMENT EATSettings (ApplicationProfiles)>
<!ATTLIST EATSettings
 ACAName CDATA #REQUIRED
>
<!ELEMENT ApplicationProfiles (Profile)*>
<!ATTLIST ApplicationProfiles
 Path CDATA #REQUIRED
>
<!ELEMENT Profile (#PCDATA)>
```

Example: eat_dresden.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE EATSettings SYSTEM "EATSettings.dtd">
<EATSettings ACAName="aca_dresden">
   <ApplicationProfiles Path="cn=appProfile,cn=eat,cn=rcl">
      <Profile>NetMeeting.xml</Profile>
      <Profile>RealSystem.xml</Profile>
      <Profile>UltimaOnline.xml</Profile>
      <Profile>UnrealTournament.xml</Profile>
      <Profile>WinSip.xml</Profile>
   </ApplicationProfiles>
</EATSettings>
```

ProxySettings.xml

```
<!ELEMENT ProxySettings (ProxyID,FullName?,ControlPort?)>
<!ATTLIST ProxySettings
 EATName CDATA #REQUIRED
>
<!ELEMENT ProxyID (#PCDATA)>
<!ELEMENT FullName (#PCDATA)>
<!ELEMENT ControlPort (#PCDATA)>
```

Example: sip_dresden.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE ProxySettings SYSTEM "ProxySettings.dtd">
<ProxySettings EATName="eat_dresden">
    <ProxyID>1</ProxyID>
    <FullName>SIP Proxy</FullName>
    <ControlPort>5004</ControlPort>
</ProxySettings>
```